The Use of Temporal Context in the Generation of Strings

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

OF THE UNIVERSITY OF STELLENBOSCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Christine du Toit

March, 2002

Supervised by: Prof. A.P.J. van der Walt

Declaration

I the undersigned hereby declare that the work contained in this thesis is my own original work and has not previously in its entirety or in part been submitted at any university for a degree.

Signature: Date:

Abstract

Grammars with regulated rewriting are used to restrict the application of context-free productions in order to avoid certain derivations. This enables these grammars to generate both context-free and non-context-free languages using only production rules with a context-free format. These grammars are more powerful than context-free grammars, but usually not as powerful as context-sensitive grammars. Various grammars with regulated rewriting have been developed and some will be discussed in this thesis.

Propositional linear temporal logic is a formal system used to describe truth values of propositions over time. This is done by defining a timeline together with a set of propositions. It is then possible to construct temporal logic formulae, consisting of these propositions and temporal operators, to specify the truth values of the propositions for every step in the timeline.

In this thesis we define and discuss temporal grammars that combine grammars with propositional linear temporal logic. Since a derivation can be associated with a timeline, a regulating device can be constructed from temporal logic formulae, that will control the application of productions within the derivation. The discussion on temporal grammars includes some of the properties of these grammars, while many ideas are illustrated by examples.

Opsomming

Grammatikas met gereguleerde herskrywing word gebruik om 'n beperking te plaas op die toepassing van konteksvrye produksies en verhoed sodoende sekere afleidings. Hierdie grammatikas beskik oor die vermoë om beide konteksvrye en nie-konteksvrye tale te genereer deur slegs produksiereëls van 'n konteksvrye formaat te gebruik. Grammatikas met gereguleerde herskrywing is dus sterker as konteksvrye grammatikas, alhoewel dit soms swakker as konteks-sensitiewe grammatikas is. 'n Verskeidenheid sulke grammatikas is al ontwikkel en sommige sal in hierdie tesis bespreek word.

Proposisionele lineêre temporale logika is 'n formele stelsel wat gebruik kan word om die waarheidswaardes van proposisies oor tyd te beskryf. Dit word gedoen deur 'n tydlyn, asook 'n versameling proposisies te definieer. Dit is dan moontlik om temporale operatore tesame met die proposisies te gebruik om temporale logika-formules te konstrueer wat in staat is om waarheidswaardes van die proposisies te spesifiseer vir elke oomblik in die tydlyn.

In hierdie tesis word temporale grammatikas, wat grammatikas met proposisionele lineêre temporale logika kombineer, gedefinieer en bespreek. Aangesien 'n afleiding met 'n tydlyn geassosieer kan word, is dit moontlik om 'n regulerende meganisme uit temporale logika-formules te konstrueer wat die toepassing van produksiereëls in die afleiding kontroleer. Die bespreking van temporale grammatikas sluit 'n verskeidenheid eienskappe van die grammatikas in, asook 'n aantal voorbeelde wat ter illustrasie gebruik word.

Acknowledgements

I gladly acknowledge the help of several people who made this project feasible:

- Prof. A.P.J. van der Walt, for introducing me to grammar theory and for his support in writing this thesis.
- Jacques Eloff, for his encouragement and interest in my work.
- My parents, for their support throughout my studies.

Contents

Chapter 1

Introduction

The theory of context-free grammars is a well-developed field of formal language theory, but according to Dassow and Păun [2] "... the world is not context-free: there is a lot of circumstances where naturally non-context-free languages appear.". They go on to state that context-sensitive grammars are "too powerful" and that the need exists for intermediate grammars that have properties similar to context-free grammars, but are more powerful than these grammars.

The idea behind all regulating mechanisms, enforced by rewriting methods, is described by Dassow, Păun and Salomaa [3] as follows: Given a context-free grammar, we restrict the application of production rules in such a way that some derivations, that are possible in the usual context-free derivation process, are avoided. This implies that the set of strings generated by a context-free grammar with a regulating device is a subset of the language generated by the same grammar without the regulating device and may therefore be a non-context-free language.

Modal logics were originally developed by philosophers to study different "modes" of truth [4]. A type of modal logic, called temporal logic, provides a formal system for reasoning about truth values of propositions that may change over time.

1.1 The Subject of this Thesis

The goal of this thesis is to examine the application of temporal logic in regulating the derivations in a grammar. Such a grammar will be known as a temporal grammar.

In the case of a temporal grammar, the regulating mechanism is a temporal formula. A derivation in the grammar is associated with a timeline and each step in the derivation is linked to a timestep. Every derived string has an n-vector assigned to it, where n is the number of variables in the grammar. This vector contains integers denoting the number of occurrences of each variable in the associated string.

The regulating device, consisting of a temporal formula, also known as a temporal restriction, involves atomic propositions (that can discriminate between n-vector values) as well as temporal operators. The regulation of the rewriting of strings is achieved by stating which atomic propositions should hold at specified timesteps. Since the propositions are associated with steps in the derivation, the temporal restrictions regulate the derivation.

1.2 An Outline of this Thesis

An overview of grammars and of propositional linear temporal logic is given in Chapter 2. In the section on propositional linear temporal logic, we discuss the linear time structure together with temporal operators.

Topics related to this thesis are discussed in Chapter 3, where we describe different control mechanisms for grammars. These include control by prescribed sequences, context conditions and partial parallelism.

Temporal grammars are defined, and their properties discussed, in Chapter 4. Some results on their generative power will be presented.

Finally, we present some conclusions in Chapter 5, where we also suggest future work concerning temporal grammars.

Chapter 2

Background

2.1 Grammars

A *grammar*, $\mathcal{G}$, is denoted by the 4-tuple (V_N, V_T, P, S) where V_N is the set of variables and V_T is the set of terminals, disjoint from V_N. P and S are the set of production rules and the start symbol respectively. A production rule may be written as a pair, (v, w), or as an expression, $v \to w$, where v and w are strings over $V_\mathcal{G} = V_N \cup V_T$. The number of symbols from V_N in v is denoted by $|v|_{V_N}$ and for any production rule $|v|_{V_N} \geq 1$.

A string $x \in V_\mathcal{G}^*$ directly *derives* $y \in V_\mathcal{G}^*$ in $\mathcal{G}$ iff $x = w_1 v w_2$ and $y = w_1 w w_2$, $w_1, w_2 \in V_\mathcal{G}^*$, and a production rule $v \to w$ exists in P. This derivation is written $x \underset{\mathcal{G}}{\Rightarrow} y$, or $x \Rightarrow y$ if $\mathcal{G}$ is clear. The yield relation, $\Rightarrow$, has a transitive and reflexive closure $\overset{*}{\Rightarrow}$, while a string w over $V_\mathcal{G}^*$ is called a *sentential form* of $\mathcal{G}$ iff $S \overset{*}{\Rightarrow} w$.

2.2 Propositional Linear Temporal Logic (PLTL)

A *linear time structure*, M, is defined (see [4], for example) as $M = (T, x, L)$, with:

- T a set of *states*;
- $x : \mathbb{N}_0 \to T$, an infinite sequence of states, the *timeline*;
- $L : T \to \text{PowerSet}(AP)$ a labelling of each state with the set of atomic propositions in AP true at that state.

A brief description of the formal syntax and semantics of PLTL will now be presented. The basic temporal operators are Fp ("eventually p"), Gp ("globally p"), Xp ("next-time p") and $p\,Uq$ ("p until q"), with p and q formulae in PLTL. These formulae are constructed using the above-mentioned operators, together with atomic propositions and the truth-functional connectives $\wedge$ (and), $\vee$ (or), $\neg$ (not), as well as $\Rightarrow$ (material implication). They are generated by the following rules:

1. every atomic proposition P is a formula;
2. if p and q are formulae, then $p \wedge q$ and $\neg p$ are formulae;
3. if p and q are formulae, then $p\,Uq$ and Xp are formulae.

The connective $p \vee q$ abbreviates $\neg(\neg p \wedge \neg q)$, while $\neg p \vee q$ is abbreviated by $\Rightarrow$ and $p \vee \neg p$ by *true*. The temporal operator Fp abbreviates *true* Up and Gp abbreviates $\neg F \neg p$.

The sequence of states $(s_i, s_{i+1}, s_{i+2}, \dots)$, also written $(x(i), x(i+1), x(i+2), \dots)$, will be denoted by x^i, while the notation $M, x \models p$ means that formula p is true for timeline $x = (s_0, s_1, s_2, \dots)$ in structure M (M is omitted if the structure is clear). It is defined inductively on the structure of the formulae, with P an atomic proposition and p and q formulae, by:

1. $x \models P$ iff $P \in L(s_0)$;
2. $x \models p \wedge q$ iff $x \models p$ and $x \models q$,
   $x \models \neg p$ iff it is not the case that $x \models p$,
   $x \models p \vee q$ iff $x \models p$ or $x \models q$;
3. $x \models (p\,Uq)$ iff $\exists j(x^j \models q$ and $\forall k < j(x^k \models p))$,
   $x \models Xp$ iff $x^1 \models p$.

The temporal operators Fp and Gp are formally defined as $x \models Fp$ iff $\exists j(x^j \models p)$ and $x \models Gp$ iff $\forall j(x^j \models p)$.

Although M has an infinite timeline, a variation of PLTL is discussed in [4] that allows for the underlying structure to be any initial segment I of $\mathbb{N}_0$ that may be finite. This can be achieved by repeating the final state of a sequence $(s_1, s_2, \dots, s_k)$ to produce $(s_1, s_2, \dots, s_k, s_k, s_k, \dots)$.

The meanings of temporal operators are adjusted to hold for timesteps in I. Since these meanings are similar to the definitions given earlier, they can informally be described as:

- $p\,U q$: q holds for some subsequent time in I and p holds until then.
- Xp: there exists a successor moment in I and p holds there.
- Fp: p holds for some subsequent time in I.
- Gp: p holds for all subsequent times in I.

Chapter 3

Related Work

Although context-free grammars are useful, there is a need for stronger grammars as, according to Dassow and Păun [2], there exist seven circumstances where context-free grammars are not sufficient. These cases include natural and programming languages as well as the language of logic. However, in some circumstances context sensitive grammars may be too powerful and a need is created for grammars that are more expressive than context-free, but not as powerful as context sensitive grammars. This is the motivation behind the use of context-free grammars together with sets of rules to govern the rewriting of strings during derivations.

Given a context-free grammar, $\mathcal{G}$, for a language, L, it is possible to apply any valid production rule to replace a variable during the derivation of strings. However, one can regulate the productions allowed in the derivation and, doing so, force the grammar to generate non-context-free languages. Many types of regulation have been developed and several of these will be discussed. Examples discussed by Dassow and Păun [2], as well as Dassow, Păun and Salomaa [3] are used to illustrate these methods of regulated rewriting.

3.1 Control by Prescribed Sequences

Prescribed sequences imply that the sequence in which production rules may be applied is defined together with the grammar. Examples of these grammars include matrix, programmed and regularly controlled grammars, that will be discussed shortly.

3.1.1 Matrix Grammars

A *matrix grammar* [1] is a quadruple $\mathcal{G} = (V_N, V_T, M, S)$ with V_N the set of variables, V_T the set of terminals and S the start symbol (these three symbols are used in all the regulating grammars for the remainder of this chapter). The matrix of rules, M, is written as a finite set of sequences, where each sequence is defined as $m : (r_1, r_2, \ldots, r_n)$, $n \geq 1$, with $r_1, r_2, \ldots, r_n$ production rules of the form $r_i : \alpha_i \rightarrow \beta_i$, $1 \leq i \leq n$. For two strings $x, y \in V_{\mathcal{G}}^*$ the derivation $x \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} y$ is permitted iff the strings $x_0, x_1, \ldots, x_n \in V_{\mathcal{G}}^*$ and a sequence $m : (r_1, r_2, \ldots, r_n) \in M$ exist such that $x_0 = x$, $x_n = y$, while $x_{i-1} = x'_{i-1}\alpha_i x''_{i-1}$ and $x_i = x'_{i-1}\beta_i x''_{i-1}$ for all $1 \leq i \leq n$ with $x'_{i-1}, x''_{i-1} \in V_{\mathcal{G}}^*$.

To illustrate matrix grammars, consider the grammar $\mathcal{G}_1 = (\{S, A, B, C\}, \{a, b, c\}, \{m_1, m_2, m_3\}, S)$, with
$m_1 : (S \rightarrow ABC)$
$m_2 : (A \rightarrow aA, B \rightarrow bB, C \rightarrow cC)$
$m_3 : (A \rightarrow a, B \rightarrow b, C \rightarrow c)$,
that generates the language $L(\mathcal{G}_1) = \{a^n b^n c^n : n \geq 1\}$.

Any derivation has to start with the use of m_1, since this is the only sequence that allows a production from the start symbol. Either m_2 or m_3 must be applied next, using one by one all the productions in the sequence. Applying m_2 p times will produce the string $a^p A b^p B c^p C$, while the application of m_3 will result in a string containing only terminals, since $m_3 : (A \rightarrow a, B \rightarrow b, C \rightarrow c)$ replaces the remaining A, B and C.

3.1.2 Programmed Grammars

A *programmed grammar*, according to Rosenkrantz [11], is a 5-tuple $\mathcal{G} = (V_T, V_N, J, P, S)$ with P a set of productions and J the set of production labels. A unique production (r, φ, ψ, V, W) is associated with each r in J, where $\varphi, \psi \in V_{\mathcal{G}}^*$ with $|\varphi|_{V_N} \geq 1$ and $V, W \subseteq J$. The production is written in the format $(r)\varphi \rightarrow \psi S(V)F(W)$ where $\varphi \rightarrow \psi \in P$. S and F denote the success field and failure field respectively. The idea behind these fields are discussed shortly.

Dassow and Păun [2] redefine a programmed grammar as being a quadruple $\mathcal{G} = (V_N, V_T, P, S)$, with P a finite set of triples of the form $(r : \alpha \rightarrow \beta, \sigma(r), \varphi(r))$, where $r : \alpha \rightarrow \beta$ is a production rule over $V_{\mathcal{G}}$, labelled by r, while $\sigma(r)$ and $\varphi(r)$ are two sets of labels of such core rules in P.

Let $\text{Lab}(P) = \{r : (r : \alpha \to \beta, \sigma(r), \varphi(r)) \in P\}$, then for (x, r_1), (y, r_2) in $V_{\mathcal{G}}^* \times \text{Lab}(P)$, $(x, r_1) \Rightarrow (y, r_2)$ iff either of the following holds:

1. $x = x_1 \alpha x_2$, $y = x_1 \beta x_2$, $x_1, x_2 \in V_{\mathcal{G}}^*$, $(r_1 : \alpha \to \beta, \sigma(r_1), \varphi(r_1)) \in P$, and $r_2 \in \sigma(r_1)$.
2. $x = y$, the rule $r_1 : \alpha \to \beta$ for some $(r_1 : \alpha \to \beta, \sigma(r_1), \varphi(r_1)) \in P$ is not applicable to x and $r_2 \in \varphi(r_1)$.

This implies that if $r_1 : \alpha \to \beta$ is effectively used, a rule contained in $\sigma(r_1)$, the success field, is used in the next step of the derivation. However, if the rule $r_1 : \alpha \to \beta$ can not be applied, a production from $\varphi(r_1)$, that is called the failure field, is used to obtain the next string.

To illustrate, consider the language $L(\mathcal{G}_2) = \{a^{2^n} : n \geq 1\}$, that can be generated by the grammar $\mathcal{G}_2 = (\{S, A\}, \{a\}, P, S)$, with P the set
$\{ \quad (r_1 : S \to AA, \{r_1\}, \{r_2, r_3\}),$
$\quad (r_2 : A \to S, \{r_2\}, \{r_1\}),$
$\quad (r_3 : A \to a, \{r_3\}, \emptyset) \quad \}.$

After applying the rule $S \to AA$, it cannot be applied again and a rule in the failure field has to be used. This field contains both r_2 and r_3 and, by applying the latter, the derivation will terminate through the repeated application of r_3. Applying r_2 will replace all instances of A with S, since this rule appears in its own success field. Suppose this derives the string S^{2^p}, $p \geq 2$. The rule r_1 now has to be applied 2^p times and doubles the length of the string to produce $A^{2^{p+1}}$. This cycle may be repeated any number of times, while the derivation can be terminated by applying r_3.

3.1.3 Regularly Controlled Grammars

A *regularly controlled grammar* [6] is a 6-tuple $\mathcal{G} = (V_N, V_T, P, S, R, F)$ with P the set of production rules, R a regular set over P and $F \subseteq P$.

The application $x \Rightarrow_p y$, with $x, y \in V_{\mathcal{G}}^*$, of a production rule $p : A \to w \in P$ is defined as follows:
$x = x_1 A x_2$ and $y = x_1 w x_2$ or
$x = y$, A does not appear in x and $p \in F$.

The language, $L(\mathcal{G})$, is generated by $\mathcal{G}$ to contain all strings derived by $S \Rightarrow_{p_1} w_1 \Rightarrow_{p_2} w_2 \Rightarrow_{p_3} \ldots \Rightarrow_{p_n} w_n = w$ where $w \in V_T^*$ and $p_1 p_2 p_3 \ldots p_n \in R$.

The following example is given as illustration:
The language $L(\mathcal{G}_3) = \{a^{2^n} : n \geq 1\}$ is generated by the regularly controlled grammar $\mathcal{G}_3 = (\{S, A, X\}, \{a\}, \{p_1, p_2, p_3, p_4, p_5\}, S, (p_1^* p_2 p_3^* p_4)^* p_5^*, \{p2, p4\})$ where

$p_1 : S \rightarrow AA$

$p_2 : S \rightarrow X$

$p_3 : A \rightarrow S$

$p_4 : A \rightarrow X$

$p_5 : S \rightarrow a$

The regular set $(p_1^* p_2 p_3^* p_4)^* p_5^*$ governs the rewriting of strings in order to produce the language $L(\mathcal{G}_3)$. Consider the string S^m, $m \geq 2$. Following the sequence of the regular set, p_1 has to be applied k times with $k \geq 0$. In the case where $k < m$, there is at least one S left in the string and p_2 is applied, as this is the next production rule in the sequence governed by R. As p_2 substitutes S with X, this will lead to a non-terminating derivation. This implies that $k = m$ has to hold in order for the derivation to terminate. Similarly, p_3 has to be applied l times where $l = 2m$. This cycle is repeated, doubling the number of S's contained in the string with each repetition, until the application of p_5 that terminates the derivation.

3.2 Control by Context Conditions

Control by *context conditions* imply that a production rule can only be applied if some specified context conditions hold.

3.2.1 Random Context Grammars

A *random context grammar* [14] is a quadruple $\mathcal{G} = (V_N, V_T, P, S)$ where P is a finite set of productions, each of the form $A \rightarrow \alpha(U;T)$. $A \rightarrow \alpha$ is a rule over $V_\mathcal{G}$, with $A \in V_N$ and $\alpha \in V_\mathcal{G}^+$. Both U and T are subsets of V_N.

The derivation $\beta A \gamma \Rightarrow \beta \alpha \gamma$, with $\beta, \gamma \in V_\mathcal{G}^*$ is allowed iff $A \rightarrow \alpha(U;T) \in P$, while every $B \in U$ and no $C \in T$ is in $\beta\gamma$. Set U is called the permitting context, while T is the forbidding context.

To illustrate, consider the grammar $\mathcal{G}_4 = (\{S, A, B, D\}, \{a\}, P, S)$ with the following random context rules:

$$P = \{\ S \to AA, (\emptyset; \{B, D\}),\quad A \to B, (\emptyset; \{S, D\}),\qquad B \to S, (\emptyset; \{A, D\}),$$
$$A \to D, (\emptyset; \{S, B\}),\quad D \to a, (\emptyset; \{S, A, B\})\ \}.$$

This grammar uses only the forbidding context and generates the language $L(\mathcal{G}_4) = \{a^{2^n} : n \geq 1\}$, as we will now show. Starting with the string S, that contains neither B or D, AA is produced. This string does not contain elements of either the set $\{S, D\}$ or $\{S, B\}$, implying that both $A \to B$ and $A \to D$ are allowed. On closer inspection it transpires that the rule $A \to D$ leads to a terminating derivation, while the application of $A \to B$ produces a string containing both A and B. The repeated application of $A \to B$ is forced until the string contains only B's. Similarly, after applying $B \to S$, any production from S is forbidden until no B's are left in the string, that results in a string containing only S's. Suppose that, after a number of iterations, the string is S^i. Applying the same sequence of rules as before, S^i will lead to A^{2i}, which in turn will lead to S^{2i}. This cycle continues until the application of $A \to D$ that leads to a terminating derivation.

3.2.2 Conditional Grammars

Conditional grammars [5] are quadruples of the form $\mathcal{G} = (V_N, V_T, P, S)$, where P is a finite set of pairs $r = (p, R)$, with p a production rule and R a regular set over $V_\mathcal{G}$.

The derivation $x \Rightarrow y$, where $x, y \in V_\mathcal{G}^*$, is allowed iff there exists a pair $r = (A \to w, R) \in P$ such that $x = x_1 A x_2$ and $y = x_1 w x_2$ with $x_1, x_2 \in V_\mathcal{G}^*$ and $x \in R$.

A conditional grammar for the language $L(\mathcal{G}_5) = \{a^{2^n} : n \geq 1\}$ is $\mathcal{G}_5 = (\{S, A\}, \{a\}, \{r_1, r_2, r_3\}, S)$ with
$r_1 = \ (S \to AA, A^*S^+)$
$r_2 = \ (A \to S, S^*A^+)$
$r_3 = \ (S \to a, a^*S^+)$.

Consider the string S^{2^n} with $n \geq 1$. Either the rule r_1 or r_3 may now be applied.

Suppose that r_3 is applied. This produces the string $S^m a S^{2^n - m - 1}$ where $m \geq 0$. However, if $m > 0$ the derivation will never terminate since none of the regular sets contain this string. This implies that $m = 0$ and the string $aS^{2^n - 1}$ is produced. Only the rule r_3 may now be applied, and after the repeated application of this rule, the string a^{2^n} is derived.

Suppose that r_1 is applied to S^{2^n}. In order to ensure that the derivation is not blocked,

the production rule is applied to the first S in the string, generating AAS^{2^n-1}. Now r_1 has to be applied repeatedly, finally producing the string $A^{2^{n+1}}$. Since only the application of r_2 is allowed at this stage of the derivation, its application is iterated until the string $S^{2^{n+1}}$ is derived.

3.2.3 Ordered Grammars

An *ordered grammar* [5] is a quadruple of the form $\mathcal{G} = (N, T, P, S)$ with P a finite (partially) ordered set of context-free productions.

If $x, y \in V_{\mathcal{G}}^*$, then $x \Rightarrow y$ iff a production $p = A \to w \in P$ exists such that $x = x_1 A x_2$ and $y = x_1 w x_2$, with $x_1, x_2 \in V_{\mathcal{G}}$, and there is no production $q = B \to v \in P$ such that $p \prec q$ and B is in x.

Consider an ordered grammar for the language $L(\mathcal{G}_6) = \{a^{2^n} : n \geq 1\}$. This grammar is given by $\mathcal{G}_6 = (\{S, A, B, D, Z\}, \{a\}, P, S)$. Figure 1 shows the partially ordered set of productions P.

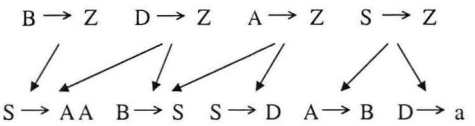

Figure 1: The Partially Ordered Set of Productions.

Since every derivation using a production of the form $X \to Z$, with $X \in \{S, A, B, D\}$, will be blocked, the following holds:

1. $S \to AA$ may only be applied if the preceding string does not contain B and D.
2. $B \to S$ may only be applied if the preceding string does not contain A and D.
3. $A \to B$ may only be applied if the preceding string does not contain S.
4. $S \to D$ may only be applied if the preceding string does not contain A.
5. $A \to a$ may only be applied if the preceding string does not contain S.

These rules are similar to the forbidding contexts listed in the random context grammar in Section 3.2.1, which implies that exactly the same derivations are allowed in both cases.

3.3 Control by Partial Parallelism

A derivation step in a context-free grammar involves the substitution of a variable by applying a production rule. In this section we discuss grammars where a derivation step consists of the parallel substitution of many variables, controlled by *partial parallelism*.

3.3.1 Indian Parallel Grammars

An *indian parallel grammar* [13] is a quadruple $\mathcal{G} = (V_N, V_T, P, S)$, where P is the set of production rules. The derivation step $x \Rightarrow y$, with $x, y \in V_{\mathcal{G}}^*$, is allowed iff the following holds:

1. $x = x_1 A x_2 A \dots x_n A x_{n+1}$, where $n \geq 0$, $A \in V_N$ and $x_i \in (V_{\mathcal{G}} - \{A\})^*$, for $1 \leq i \leq n+1$;
2. $y = x_1 w x_2 w \dots x_n w x_{n+1}$;
3. $A \to w \in P$.

The grammar $\mathcal{G}_7 = (\{S, A\}, \{a, b\}, P, S)$, with

$$P = \{\ S \to AA,\ A \to aA,\ A \to bA,\ A \to a,\ A \to b\ \},$$

is used to illustrate indian parallel grammars.

Consider the string $wAwA$ (the first step of the derivation generates AA that is of this form). By applying the productions with left side A, the following derivations are obtained:

$wAwA \Rightarrow waAwaA$

$wAwA \Rightarrow wbAwbA$

$wAwA \Rightarrow wawa$

$wAwA \Rightarrow wbwb$

Clearly, $L(\mathcal{G}_7) = \{ww : w \in \{a, b\}^*\}$.

3.3.2 k-Grammars

A *k-grammar* [8, 9] is a quadruple of the form $\mathcal{G} = (V_N, V_T, P, S)$, where P is the set of production rules and $k \geq 1$. The derivation $x \Rightarrow y$, with $x, y \in V_{\mathcal{G}}^*$, is allowed iff $x = S$ and $S \to y \in P$ or the following conditions hold:

1. $x = x_1 A_1 x_2 A_2 \ldots x_k A_k x_{k+1}$, with $x_1, x_2, \ldots, x_{k+1} \in V_{\mathcal{G}}^*$;
2. $y = x_1 w_1 x_2 w_2 \ldots x_k w_k x_{k+1}$;
3. $A_1 \to w_1, A_2 \to w_2, \ldots, A_k \to w_k \in P$.

To illustrate, consider $\mathcal{G}_8 = (\{S, A, B, C\}, \{a, b, c\}, P, S)$ with P a set containing the following production rules:

$\{ \quad S \to ABC,$
$\quad A \to aA|a,$
$\quad B \to bB|b,$
$\quad C \to cC|c \ \}.$

Suppose $\mathcal{G}_8$ is a 2-grammar. This implies that every step of the derivation, excluding the first one, should replace two variables in the string. The first step is $S \Rightarrow ABC$, while every subsequent step will add two occurrences of terminals to the string. The language generated by this grammar is $\{a^n b^m c^r : n + m + r = 2t, t \geq 1\}$.

To generate the language $\{a^n b^n c^n : n \geq 1\}$, $\mathcal{G}_8$ is considered a 3-grammar. Since three variables have to be replaced during any step (excluding the first), the diagram below illustrates all possible derivations.

$$\begin{array}{ccccccccccc} S & \Rightarrow & ABC & \Rightarrow & aAbBcC & \Rightarrow & \ldots & \Rightarrow & a^n A b^n B c^n C & \Rightarrow & \ldots \\ & & \Downarrow & & \Downarrow & & & & \Downarrow & & \\ & & abc & & a^2 b^2 c^2 & & & & a^{n+1} b^{n+1} c^{n+1} & & \end{array}$$

The first step of the derivation produces ABC. After that the derivation becomes blocked whenever the string does not contain at least three variables to replace.

For the k-grammar $\mathcal{G}_8$ with $k \geq 4$, the derivation becomes blocked immediately after deriving ABC from S, since ABC does not contain 4 variables.

3.3.3 Scattered Context Grammars

A *scattered context* grammar [7] is a quadruple $\mathcal{G} = (V_N, V_T, P, S)$ where P is a finite set of matrices of the form $(A_1 \to w_1, A_2 \to w_2, \ldots, A_k \to w_k)$ with $k \geq 0$, $A_j \in V_N$

and $w_j \in V_{\mathcal{G}}$ where $1 \leq j \leq k$.

For $x, y \in V_{\mathcal{G}}^*$, $x \Rightarrow y$ is allowed, iff the following conditions hold:

1. $x = x_1 A_1 x_2 A_2 \dots x_k A_k x_{k+1}$, for $x_1, x_2, \dots, x_{k+1} \in V_{\mathcal{G}}^*$;
2. $y = x_1 w_1 x_2 w_2 \dots x_k w_k x_{k+1}$;
3. $(A_1 \to w_1, A_2 \to w_2, \dots, A_k \to w_k) \in P$.

Consider the scattered context grammar $\mathcal{G}_9 = (\{S, A\}, \{a, b, c\}, P, S)$, where $P = \{(S \to AAA), (A \to aA, A \to bA, A \to cA), (A \to a, A \to b, A \to c)\}$.

Examine the following diagram:

$$\begin{array}{ccccccccc} S & \Rightarrow & AAA & \Rightarrow & aAbAcA & \Rightarrow & \dots & \Rightarrow & a^n A b^n A c^n A & \Rightarrow & \dots \\ & & \Downarrow & & \Downarrow & & & & \Downarrow & & \\ & & abc & & a^2 b^2 c^2 & & & & a^{n+1} b^{n+1} c^{n+1} & & \end{array}$$

The diagram illustrates all the possible derivations in $\mathcal{G}_9$, showing that the grammar produces the language $L(\mathcal{G}_9) = \{a^n b^n c^n : n \geq 1\}$.

Chapter 4

Temporal Grammars

4.1 Temporal Context

An examination of the grammars with regulated rewriting highlighted in Chapter 3 shows that they all achieve their aim by somehow prescribing which production should be attempted next in the derivation. One point of view that seems to be absent, is that of considering a derivation to be a process that evolves over time. Taking this approach, one is led quite naturally to the idea of employing some kind of temporal logic to set up regulatory devices for the way in which a derivation may evolve. Before formulating this notion, we present an easy example to explain the main issues surrounding this approach.

4.1.1 Restricting the Parikh Vector Entries for Derived Strings

Consider the context-free grammar $\mathcal{G} = (V_N, V_T, P, S)$, with $V_N = \{S, A, B, C\}$, $V_T = \{a\}$ and

$$P = \{ \; S \to A, \quad A \to B|a, \quad B \to C, \quad C \to SS \; \},$$

which generates the language $L(\mathcal{G}) = \{a^*\}$. A typical derivation is:
$S \Rightarrow A \Rightarrow B \Rightarrow C \Rightarrow SS \Rightarrow AS \Rightarrow AA \Rightarrow BA \Rightarrow CA \Rightarrow SSA \Rightarrow ASA \Rightarrow AAA \Rightarrow aAA \Rightarrow aaA \Rightarrow aaa.$

Partial information about this derivation is captured in the sequence of Parikh vectors,

$$\begin{array}{lllllllll}
(1,0,0,0,0) & \Rightarrow & (0,1,0,0,0) & \Rightarrow & (0,0,1,0,0) & \Rightarrow & (0,0,0,1,0) & \Rightarrow \\
(2,0,0,0,0) & \Rightarrow & (1,1,0,0,0) & \Rightarrow & (0,2,0,0,0) & \Rightarrow & (0,1,1,0,0) & \Rightarrow \\
(0,1,0,1,0) & \Rightarrow & (2,1,0,0,0) & \Rightarrow & (1,2,0,0,0) & \Rightarrow & (0,3,0,0,0) & \Rightarrow \\
(0,2,0,0,1) & \Rightarrow & (0,1,0,0,2) & \Rightarrow & (0,0,0,0,3), & & & \\
\end{array}$$

where we have adopted the ordering $S \prec A \prec B \prec C \prec a$, and the *Parikh vector* of a word in $V_{\mathcal{G}}^*$ gives the number of occurrences of the symbols in this ordering.

In order to regulate the grammar, we want to restrict the application of production rules in such a way that some derivations are avoided. Since the Parikh vectors denote the number of occurrences of each symbol in every derived string, it is possible to regulate the generation of these strings by placing restrictions on the vector entries. For a derivation in $\mathcal{G}$, suppose those vector entries associated with the variables S and B may not simultaneously be larger than 0. Let the same hold for the entries associated with A and C, as well as those associated with a and B. This forces any derivation to avoid strings containing both S and B, A and C, or a and B. Clearly, the derivation discussed earlier violates this restriction, since the string CA is produced.

To show which derivations are still allowed, consider the string S^i, $i \geq 1$, with which we associate the Parikh vector $(i,0,0,0,0)$. The only rule allowing a production from S is $S \to A$, which produces the string $S^j A S^{i-j-1}$, $0 \leq j \leq i-1$, with vector $(i-1,1,0,0,0)$. If $i > 1$, there is at least one occurrence of S in $S^j A S^{i-j-1}$, implying that $A \to B$ may not be applied at this time and forcing the derivation to replace every occurrence of S before allowing B in the string. However, the production $A \to a$ is still allowed and, as long as B does not appear in the string, this may be applied at any time. Let us consider this case. As soon as a is introduced to the string, A may never be replaced by B, as this will allow the vector entries for both B and a to become non-zero. Therefore every A that was produced by S will be replaced by a, and finally produce a^i, which concludes the derivation.

If we do not produce any a's, an instance of A in A^i, with vector $(0,i,0,0,0)$, may be replaced by B. After doing this, the production $A \to a$ is not allowed anymore and, since C may not appear in the string together with A, all instances of A have to be replaced by B. This derives the string B^i, with which we associate the vector $(0,0,i,0,0)$. The string C^i is derived in a similar fashion, since every B is replaced by C, while $C \to SS$ may not be applied until the vector entry for B is 0.

Finally, $C \to SS$ is applied. At this stage the application of the rule $S \to A$ derives an

illegal string, and the derivation is forced to replace all instances of C. The string S^{2i} is derived, which resets the cycle. Clearly, the language generated by this grammar is $\{a^{2^n} : n \geq 0\}$, which is not context-free.

In order to simplify the association of a grammar with a temporal structure, we eliminate the need to map terminals to the Parikh vector. This is achieved by importing a new variable to replace every instance of a terminal symbol that is currently used in at least one of the regulating rules. A production rule is added to allow the new variable to produce the terminal.

To illustrate, consider the set of production rules in $\mathcal{G}$

$$\begin{aligned} P = \ \{ \ & S \to A, \\ & A \to B|a, \\ & B \to C, \\ & C \to SS \ \}. \end{aligned}$$

Since the terminal a is used in a regulating rule, we need to replace it with a new variable, V, and change P accordingly:

$$\begin{aligned} P = \ \{ \ & S \to A, \\ & A \to B|V, \\ & B \to C, \\ & C \to SS, \\ & V \to a \ \}. \end{aligned}$$

The rule forbidding both the vector entries associated with B and a to be non-zero now applies to B and V.

We will now formulate the association of grammars with temporal structures. From the example it should be clear that we need to link a derivation in $\mathcal{G}$ with a sequence of vectors similar to Parikh vectors. Furthermore, we require a way to stipulate the restrictions on these vectors, for which we will use temporal logic.

4.1.2 The Association of Grammars with Temporal Structures

Let $\mathcal{G} = (V_N, V_T, P, S)$, with $V_N = \{S_0, S_1, \ldots, S_n\}$ and $S = S_0$, be a context-free grammar, and suppose $\Gamma : S_0 = w_0 \underset{\mathcal{G}}{\Rightarrow} w_1 \underset{\mathcal{G}}{\Rightarrow} w_2 \underset{\mathcal{G}}{\Rightarrow} \ldots \underset{\mathcal{G}}{\Rightarrow} w_k = \alpha$, with $\alpha \in V_T^*$, is a derivation.

We want to show how to associate a linear time structure $M = M(\Gamma) = (T, x, L)$, defined in Section 2.2, with Γ. To this end we allow M to have a finite timeline, as

discussed in Section 2.2, and define

- T, the set of states, by $T = (\mathbb{N}_0)^{n+1}$.
- $x(i) = (v_0^i, v_1^i, v_2^i, \ldots, v_n^i)$, $i \geq 0$, is a finite sequence of states, where v_j^i is the number of occurrences of variable S_j in w_i. This is similar to the Parikh vector, but only the variables are mapped. For the remainder of this thesis the vector will be known as the n-vector associated with w_i.
- $AP = \{PZ_i | 0 \leq i \leq n\} \cup \{PG_i | 0 \leq i \leq n\}$. Here PZ_j is true for the vector $(v_0, v_1, v_2, \ldots, v_n)$ iff $v_j = 0$, and PG_k is true iff $v_k > 0$.
- $L : T \rightarrow \text{PowerSet}(AP)$ is given as $L((v_0, v_1, v_2, \ldots, v_n)) = \{PZ_i | PZ_i \text{ is true at } (v_0, v_1, v_2, \ldots, v_n)\} \cup \{PG_i | PG_i \text{ is true at } (v_0, v_1, v_2, \ldots, v_n)\}$.

M can be used to discriminate between strings in $L(\mathcal{G})$ by stipulating a temporal logic formula that must hold for the linear time structure associated with at least one derivation of a string.

Before continuing, we present some of the notation used in the remainder of this thesis.

The symbol G will be used to specify the "globally" modality in PLTL, while $\mathcal{G}$ is used to denote a grammar. Lower case letters near the end of the alphabet are used to represent sentential forms, while lower case Greek letters denote strings in V_T^*. Upper case Greek letters are used to denote full or partial derivations. Such a derivation will be written in the form $\Gamma : w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_k$, where $w_1, w_2, \ldots w_k$ are sentential forms.

Variables in $\mathcal{G}$ will usually be denoted by S_i, for some appropriate range of index i. Subscripts of variables will sometimes be chosen to facilitate understanding. For example, a variable producing the terminal a may be written S_a.

In order to illustrate the use of the linear time structure, again consider the context-free grammar $\mathcal{G} = (V_N, V_T, P, S)$ used at the beginning of this chapter. The production rules

$$\begin{aligned} P = \{ \quad & S \rightarrow A, \\ & A \rightarrow B|V, \\ & B \rightarrow C, \\ & C \rightarrow SS, \\ & V \rightarrow a \ \} \end{aligned}$$

are rewritten as

$$\begin{aligned} P = \{\ & S_1 \to S_2, \\ & S_2 \to S_3|S_5, \\ & S_3 \to S_4, \\ & S_4 \to S_1S_1, \\ & S_5 \to a\ \}, \end{aligned}$$

while the set of variables and the start symbol change accordingly:

$V_N = \{S_1, S_2, S_3, S_4, S_5\}$ and

$S = S_1$.

We associate the linear time structure $M(\Gamma)$ with a derivation Γ in $\mathcal{G}$. In this case $T = (\mathrm{N}_0)^5$, while $x(i) = (v_1^i, v_2^i, v_3^i, v_4^i, v_5^i)$, $i \geq 0$, where v_j^i is the number of occurrences of S_j in the ith derived string. $AP = \{PZ_i|1 \leq i \leq 5\} \cup \{PG_i|1 \leq i \leq 5\}$ and $L((v_1, v_2, v_3, v_4, v_5)) = \{PZ_i|PZ_i$ is true at $(v_1, v_2, v_3, v_4, v_5)\} \cup \{PG_i|PG_i$ is true at $(v_1, v_2, v_3, v_4, v_5)\}$.

We now construct a temporal formula to regulate the application of productions: $M, x \models G\neg[(PG_1 \wedge PG_3) \vee (PG_2 \wedge PG_4) \vee (PG_3 \wedge PG_5)]$, which implies that "globally it is not true that both PG_1 and PG_3, or PG_2 and PG_4, or PG_3 and PG_5 are true", where PG_1, PG_2, PG_3, PG_4 and PG_5 are respectively true if $v_1 > 0$, $v_2 > 0$, $v_3 > 0$, $v_4 > 0$ and $v_5 > 0$.

This restriction allows similar derivations as discussed earlier in this chapter, and produces $\{a^{2^n} : n \geq 0\}$.

As mentioned earlier, a context-free grammar associated with a linear time structure is called a *temporal grammar*. The definition of such a grammar is formulated in Section 4.1.3.

4.1.3 The Definition of Temporal Grammars

A *temporal grammar* is a 5-tuple $\mathcal{G} = \{V_N, V_T, P, S, q\}$ where

1. V_N is the set of variables;
2. V_T is the set of terminals, disjoint from V_N;
3. $S \in V_N$ is the start symbol;
4. P is the set of context-free production rules, where $S \to \epsilon$ (with ϵ the empty

string) is only allowed if S is the start symbol and if S does not occur on the right hand side of any production rule;

5. q is a temporal logic formula.

The *language* $L(\mathcal{G})$, generated by $\mathcal{G}$, consists of all the strings $w \in V_T^*$ such that there is a derivation $\Gamma : S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n = w$, where no step in the derivation violates the temporal restriction, q, on the linear time structure, M, associated with Γ.

The use of the symbols PG and PZ in the temporal formula is not intuitively appealing. We have the option to use the n-vector entries instead, and respectively replacing PG_j and PZ_j with $v_j > 0$ and $v_j = 0$. However, these entries do not improve the appeal of the formula and prompts us to abuse the notation. For the remainder of this thesis we will use the variable symbols in temporal formulae, and will write $S_j > 0$ and $S_j = 0$ instead of PG_j and PZ_j respectively. We also omit the use of $M, x \models$, since this is clear for all formulae.

We will now present two examples of temporal grammars in order to illustrate the use of these grammars in the generation of non-context-free languages.

4.1.4 Examples of Temporal Grammars

The temporal logic formula for the first example contains many instances of the X-operator (nexttime operator). Due to this, the formula $\underbrace{XX\ldots X}_{n}(p)$, with p a formula, will be written $X^n(p)$.

Consider the temporal grammar $\mathcal{G} = (V_N, V_T, P, S_0, q)$, with

$V_N = \{S_0, S_1, S_2, S_3, S_a, S_b, S_c\}$,

$V_T = \{a, b, c\}$,

$P = \{ \begin{array}{lcl} S_0 & \rightarrow & S_1S_2S_3, \\ S_1 & \rightarrow & S_aS_1|a, \\ S_2 & \rightarrow & S_bS_2|b, \\ S_3 & \rightarrow & S_cS_3|c, \\ S_a & \rightarrow & a, \\ S_b & \rightarrow & b, \\ S_c & \rightarrow & c \ \} \text{ and} \end{array}$

$$q = G(\ [(S_1 > 0) \wedge (S_2 > 0) \wedge (S_3 > 0) \wedge (S_a = 0) \wedge (S_b = 0) \wedge (S_c = 0)] \Rightarrow$$
$$[X(S_a > 0) \wedge X^2(S_b > 0) \wedge X^3(S_c > 0) \wedge$$
$$X^4(S_a = 0) \wedge X^5(S_b = 0) \wedge X^6(S_c = 0)] \vee$$
$$[X(S_1 = 0) \wedge X^2(S_2 = 0) \wedge X^3(S_3 = 0)]\ ).$$

Consider the string $a^m S_1 b^m S_2 c^m S_3$, $m \geq 0$. For this string, the propositions $S_1 > 0$, $S_2 > 0$, $S_3 > 0$, $S_a = 0$, $S_b = 0$ and $S_c = 0$ are all true, and according to the temporal restriction one of the following is implied:

- $X(S_a > 0) \wedge X^2(S_b > 0) \wedge X^3(S_c > 0) \wedge X^4(S_a = 0) \wedge X^5(S_b = 0) \wedge X^6(S_c = 0)$, that derives $a^{m+1} S_1 b^{m+1} S_2 c^{m+1} S_3$ from $a^m S_1 b^m S_2 c^m S_3$:
  $X(S_a > 0)$ requires S_a to appear in the next string, and $a^m S_a S_1 b^m S_2 c^m S_3$ is derived. Following this, $X^2(S_b > 0)$ forces the production of $a^m S_a S_1 b^m S_b S_2 c^m S_3$. $X^3(S_c > 0)$ needs to hold at the next timestep, and $a^m S_a S_1 b^m S_b S_2 c^m S_c S_3$ is derived, after which $X^4(S_a = 0)$ forces the elimination of S_a. This is achieved by applying the production $S_a \rightarrow a$ and deriving the string $a^{m+1} S_1 b^m S_b S_2 c^m S_c S_3$. In a similar fashion, $X^5(S_b = 0)$ forces the application of $S_b \rightarrow b$ to derive $a^{m+1} S_1 b^{m+1} S_2 c^m S_c S_3$. Finally, $X^6(S_c = 0)$ requires S_c to be removed from the string. This is done by applying the production $S_c \rightarrow c$ and deriving $a^{m+1} S_1 b^{m+1} S_2 c^{m+1} S_3$, which resets the cycle.

- $X(S_1 = 0) \wedge X^2(S_2 = 0) \wedge X^3(S_3 = 0)$, that terminates the derivation:
  $X(S_1 = 0)$ forces the elimination of S_1 from $a^m S_1 b^m S_2 c^m S_3$. This is achieved through the application of the production $S_1 \rightarrow a$, that yields $a^{m+1} b^m S_2 c^m S_3$. Similarly, $X^2(S_2 = 0)$ requires S_2 to be removed from the string. The production rule $S_2 \rightarrow b$ is applied and $a^{m+1} b^{m+1} c^m S_3$ is derived. Finally, $X^3(S_3 = 0)$ ends the derivation by forcing S_3 to be removed. The resulting string is $a^{m+1} b^{m+1} c^{m+1}$ and the derivation terminates.

Clearly, $\mathcal{G}$ generates the non-context-free language $\{a^r b^r c^r : r \geq 1\}$.

The second example is the grammar $\mathcal{G} = (V_N, V_T, P, S_0, q)$, with
$V_N = \{S_0, S_1, S_2, S_3, S_5, S_4, S_6, S_a, S_b\}$,
$V_T = \{a, b\}$,

$$
\begin{aligned}
P = \{\ & S_0 \rightarrow S_1S_2, \\
& S_1 \rightarrow S_3S_1|S_5S_1|S_3|S_5, \\
& S_2 \rightarrow S_4S_2|S_6S_2|S_4|S_6, \\
& S_3 \rightarrow a, \\
& S_4 \rightarrow a, \\
& S_5 \rightarrow b, \\
& S_6 \rightarrow b\ \} \text{ and}
\end{aligned}
$$

$$
\begin{aligned}
q = G(\ & [(S_3 > 0) \wedge (S_4 = 0) \Rightarrow X(S_4 > 0)] \wedge \\
& [(S_5 > 0) \wedge (S_6 = 0) \Rightarrow X(S_6 > 0)] \wedge \\
& [(S_4 > 0) \wedge (S_3 = 0) \Rightarrow X(S_3 > 0)] \wedge \\
& [(S_6 > 0) \wedge (S_5 = 0) \Rightarrow X(S_5 > 0)] \wedge \\
& [(S_3 > 0) \wedge (S_4 > 0) \Rightarrow X(S_3 = 0) \wedge XX(S_4 = 0)] \wedge \\
& [(S_5 > 0) \wedge (S_6 > 0) \Rightarrow X(S_5 = 0) \wedge XX(S_6 = 0)]\ ).
\end{aligned}
$$

Starting with the variable S_0, the production $S_0 \rightarrow S_1S_2$ is applied and the string S_1S_2 is produced. Applying another production rule results in one of the following strings: $S_3S_1S_2$, $S_5S_1S_2$, S_3S_2, S_5S_2, $S_1S_4S_2$, $S_1S_6S_2$, S_1S_4 or S_1S_6. Depending on the respective variables S_3, S_4, S_5 or S_6 in the current string, the temporal restriction requires the next string to contain S_4, S_3, S_6 or S_5. If S_1 (S_2) is absent in the new string, while S_2 (S_1) is present, the derivation will fail, since the application of any sequence of productions will eventually violate the temporal restriction. In the case where neither S_1 nor S_2 appears in the string, the length of the string will not increase and the derivation will terminate, producing aa or bb.

In the case where both S_1 and S_2 are still contained in the string, one of the pairs S_3 and S_4, or S_5 and S_6 will also be present. During the next two steps of the derivation, either S_3 is removed, followed by S_4, or S_5 is removed, followed by S_6. This is achieved by substituting these variables with a or b. The string now has the form $wS_1wS_2, w \in \{a, b\}$. Clearly the grammar generates the language $\{ww : w \in \{a, b\}^*\}$, since any string of the form $w_1w_2S_1w_1w_2S_2$, with $w_1 \in \{a, b\}^*$ and $w_2 \in \{a, b\}$, can be derived from $w_1S_1w_1S_2$ by repeating the cycle.

4.1.5 Frequently Used Constructions

A temporal restriction consists of one temporal formula that is true for the entire timeline. Suppose this formula is of the form $G[p_1 \wedge p_2 \wedge \cdots \wedge p_m]$, $m \geq 0$, then every *clause* $p_j, 1 \leq j \leq m$ in this formula has to hold during the entire derivation. A

temporal grammar has a finite set of production rules and, if the temporal restriction is of the above-mentioned form, it also has a finite number of clauses. This implies that in many cases some production rules will be used more than once when deriving long strings, while a part of the sequence in which productions are applied will be used repeatedly. In this section we will discuss two general constructions for grammars generating certain types of languages.

Increasing Multiples of a Single Variable

Quite often, the number of variables in a string have to be increased in such a way that every instance of a variable is replicated a certain number of times. For example, starting with the string a, the language $\{a^{2^p} : p \geq 0\}$ requires the number of a's contained in the string to double to aa, then each a has to be doubled again, producing $aaaa$, etc. This example shows a general way for creating multiples of a single variable.

Suppose $\mathcal{G} = (V_N, V_T, P, S_0, q)$ is a temporal grammar with

$V_N = \{S_0, S_1, \ldots, S_m\}$,

$V_T = \{a\}$,

$$
\begin{aligned}
P = \{\ & S_0 && \rightarrow && S_1^{k_1}, \\
& S_1 && \rightarrow && S_2^{k_2}, \\
& S_2 && \rightarrow && S_3^{k_3}, \\
& \vdots \\
& S_{m-1} && \rightarrow && S_m^{k_m}, \\
& S_m && \rightarrow && S_0 | S_a, \\
& S_a && \rightarrow && a\ \},
\end{aligned}
$$

where $S_r^{k_r}$ is a string consisting of k_r instances of the variable S_r, and

$$
\begin{aligned}
q = G(\ & [(S_0 > 0) && \Rightarrow && X(S_2 = 0)U(S_0 = 0)] \wedge \\
& [(S_1 > 0) && \Rightarrow && X(S_3 = 0)U(S_1 = 0)] \wedge \\
& [(S_2 > 0) && \Rightarrow && X(S_4 = 0)U(S_2 = 0)] \wedge \\
& \vdots \\
& [(S_{m-1} > 0) && \Rightarrow && X(S_0 = 0)U(S_{m-1} = 0)] \wedge \\
& [(S_m > 0) && \Rightarrow && X(S_1 = 0)U(S_m = 0)] \wedge \\
& [\neg((S_a > 0) \wedge (S_0 > 0))]\ ).
\end{aligned}
$$

The following strings are generated by the grammar:

$$
\begin{aligned}
S_0 &\Rightarrow S_1^{k_1} \\
&\overset{*}{\Rightarrow} S_2^{k_1 \times k_2} \\
&\vdots \\
&\overset{*}{\Rightarrow} S_m^{k_1 \times k_2 \times \cdots \times k_m} \\
&\overset{*}{\Rightarrow} S_0^{k_1 \times k_2 \times \cdots \times k_m} && (\text{or } a^{k_1 \times k_2 \times \cdots \times k_m}) \\
&\overset{*}{\Rightarrow} S_1^{(k_1 \times k_2 \times \cdots \times k_m) k_1} \\
&\overset{*}{\Rightarrow} S_2^{(k_1 \times k_2 \times \cdots \times k_m) k_1 \times k_2} \\
&\vdots \\
&\overset{*}{\Rightarrow} S_m^{(k_1 \times k_2 \times \cdots \times k_m) k_1 \times k_2 \times \cdots \times k_m} \\
&\overset{*}{\Rightarrow} S_0^{(k_1 \times k_2 \times \cdots \times k_m)^2} && (\text{or } a^{(k_1 \times k_2 \times \cdots \times k_m)^2}) \\
&\vdots \\
&\overset{*}{\Rightarrow} S_0^{(k_1 \times k_2 \times \cdots \times k_m)^l} && (\text{or } a^{(k_1 \times k_2 \times \cdots \times k_m)^l}) \\
&\overset{*}{\Rightarrow} S_1^{(k_1 \times k_2 \times \cdots \times k_m)^l k_1} \\
&\overset{*}{\Rightarrow} S_2^{(k_1 \times k_2 \times \cdots \times k_m)^l k_1 \times k_2} \\
&\vdots \\
&\overset{*}{\Rightarrow} S_m^{(k_1 \times k_2 \times \cdots \times k_m)^l k_1 \times k_2 \times \cdots \times k_m} \\
&\overset{*}{\Rightarrow} S_0^{(k_1 \times k_2 \times \cdots \times k_m)^{l+1}} && (\text{or } a^{(k_1 \times k_2 \times \cdots \times k_m)^{l+1}}) \\
&\vdots
\end{aligned}
$$

Increasing Variables in Equivalent Numbers

Some languages require a different method for generating strings with increasing length. In this case more than one terminal is involved, all of which should be present in the string in equal numbers. An example of such a language is $\{a^p b^p c^p | p \geq 1\}$. We will deviate from our usual notation to simplify keeping track of the derivation. In this case the variables $S_0, S_1, S_2, \ldots, S_m$ are respectively associated with $S_{1\bullet}, S_{2\bullet}, \ldots, S_{m\bullet}$, that will facilitate the correct increase in the number of occurrences of every symbol.

Let $\mathcal{G} = (V_N, V_T, P, S_0, q)$ be a context-free grammar, with
$V_N = \{S_0, S_1, S_2, \ldots, S_m, S_{1\bullet}, S_{2\bullet}, \ldots, S_{m\bullet}, S_{a_1}, S_{a_2}, \ldots, S_{a_m}, S_{a_1}, S_{a_2}, \ldots, S_{a_m}\}$,
$V_T = \{a_1, a_2, \ldots, a_m\}$,

$$
\begin{aligned}
P = \{\ & S_0 \rightarrow S_1 S_2 ... S_m, \\
& S_1 \rightarrow S_{1\bullet}, \\
& S_2 \rightarrow S_{2\bullet}, \\
& \vdots \\
& S_{m-1} \rightarrow S_{m-1\bullet}, \\
& S_m \rightarrow S_{m\bullet}, \\
& S_{1\bullet} \rightarrow a_1 S_1 | S_{a_1}, \\
& S_{2\bullet} \rightarrow a_2 S_2 | S_{a_2}, \\
& \vdots \\
& S_{m-1\bullet} \rightarrow a_{m-1} S_{m-1} | S_{a_{m-1}}, \\
& S_{m\bullet} \rightarrow a_m S_m | S_{a_m}, \\
& S_{a_1} \rightarrow a_1, \\
& S_{a_2} \rightarrow a_2, \\
& \vdots \\
& S_{a_{m-1}} \rightarrow a_{m-1}, \\
& S_{a_m} \rightarrow a_m\ \} \text{ and}
\end{aligned}
$$

$$
\begin{aligned}
q = G(\ & [(S_{1\bullet} = 0) \wedge (S_{2\bullet} = 0) \wedge \cdots \wedge (S_{m-1\bullet} = 0) \wedge (S_{m\bullet} = 0) \Rightarrow \\
& X(S_1 = 0) \wedge XX(S_2 = 0) \wedge \cdots \wedge \underbrace{XX \ldots X}_{m-1}(S_{m-1} = 0) \wedge \underbrace{XX \ldots X}_{m}(S_m = 0)] \ \wedge \\
& [(S_{1\bullet} > 0) \wedge (S_{2\bullet} > 0) \wedge \cdots \wedge (S_{m-1\bullet} > 0) \wedge (S_{m\bullet} > 0) \Rightarrow \\
& (X(S_1 > 0) \wedge XX(S_2 > 0) \wedge \cdots \wedge \underbrace{XX \ldots X}_{m-1}(S_{m-1} > 0) \wedge \underbrace{XX \ldots X}_{m}(S_m > 0)) \ \vee \\
& (X(S_{a_1} > 0) \wedge XX(S_{a_2} > 0) \wedge \cdots \wedge \underbrace{XX \ldots X}_{m-1}(S_{a_{m-1}} > 0) \wedge \underbrace{XX \ldots X}_{m}(S_{a_m} > 0) \\
& \wedge (X(S_1 = 0) \wedge XX(S_2 = 0) \wedge \cdots \wedge \underbrace{XX \ldots X}_{m-1}(S_{m-1} = 0) \wedge \underbrace{XX \ldots X}_{m}(S_m = 0))] \ ).
\end{aligned}
$$

The grammar generates the following strings:

$$
\begin{aligned}
S_0 &\Rightarrow S_1 S_2 S_3 \ldots S_{m-1} S_m \\
&\Rightarrow S_{1\bullet} S_2 S_3 \ldots S_{m-1} S_m \\
&\Rightarrow S_{1\bullet} S_{2\bullet} S_3 \ldots S_{m-1} S_m \\
&\vdots \\
&\Rightarrow S_{1\bullet} S_{2\bullet} S_{3\bullet} \ldots S_{m-1\bullet} S_m \\
&\Rightarrow S_{1\bullet} S_{2\bullet} S_{3\bullet} \ldots S_{m-1\bullet} S_{m\bullet} \\
&\Rightarrow a_1 S_1 S_{2\bullet} S_{3\bullet} \ldots S_{m-1\bullet} S_{m\bullet} \\
&\Rightarrow a_1 S_1 a_2 S_2 S_{3\bullet} \ldots S_{m-1\bullet} S_{m\bullet} \\
&\vdots
\end{aligned}
$$

$$
\begin{aligned}
&\Rightarrow\ a_1S_1a_2S_2a_3S_3\ldots a_{m-1}S_{m-1}S_m\bullet\\
&\Rightarrow\ a_1S_1a_2S_2a_3S_3\ldots a_{m-1}S_{m-1}a_mS_m\\
&\vdots\\
&\Rightarrow\ (a_1)^{p-1}S_1(a_2)^{p-1}S_2(a_3)^{p-1}S_3\ldots(a_{m-1})^{p-1}S_{m-1}(a_m)^{p-1}S_m\\
&\overset{*}{\Rightarrow}\ (a_1)^p(a_2)^p(a_3)^p\ldots(a_{m-1})^p(a_m)^p,
\end{aligned}
$$

with $p \geq 1$.

4.2 The Generative Power of Temporal Grammars

In this section we establish some results that give an indication of the generative power of temporal grammars. First, in Section 4.2.1, we show that temporal grammars are at least as powerful as random context grammars. This implies that temporal grammars are also at least as powerful as matrix grammars, since an equivalence between matrix and random-context grammars was established by Mayer [10] in 1972. An equivalence between matrix and programmed grammars was proved in 1970 by A. Salomaa [12], which implies that temporal grammars are also at least as powerful as programmed grammars.

Next, in Section 4.2.2, we give some evidence to support the conjecture that temporal grammars, in fact, may be stronger than random context grammars. Lastly, in Section 4.2.3, we offer some views on the generative power of temporal grammars.

4.2.1 A Comparison with Random Context Grammars

When examining the regulating device for random context grammars (see Section 3.2.1), it is clear that it depends on the appearance, or lack thereof, of certain symbols in a sentential form. This can be simulated by temporal grammars, since the atomic proposition $PG_1 = (S_1 > 0)$ implies that S_1 appears in the string, while $PZ_1 = (S_1 = 0)$ implies the absence of S_1. To simulate a production rule $A \to w\ (U;T)$ in the random context grammar, the temporal grammar needs to apply the production $A \to w$ if the elements in U are present in the current string, while the elements in T are absent. To achieve this, a new variable B is added to the set of variables in the temporal grammar, while the productions $A \to B$ and $B \to w$ are added to the set of production rules. Should the elements of U appear in a string, while the elements of T are absent, the temporal grammar requires B to be in the next string, but not in the string after that.

This forces the application $A \to B$, followed by $B \to w$, that simulates the application of $A \to w$ in the random context grammar.

To illustrate, consider the grammar $\mathcal{G}_1 = (\{S, A, B, D\}, \{a\}, P_1, S)$ with the following random context rules:

$P_1 = \{ \quad S \to AA, (\emptyset; \{B, D\}), \quad A \to B, (\emptyset; \{S, D\}), \qquad B \to S, (\emptyset; \{A, D\}),$
$A \to D, (\emptyset; \{S, B\}), \quad D \to a, (\emptyset; \{S, A, B\}) \quad \}.$

This grammar was discussed in Section 3.2.1, and generates the language $\{a^{2^n} : n \geq 1\}$.

We convert this grammar to a temporal grammar, $\mathcal{G}_2 = (V_N, \{a\}, P_2, S, q)$, by first adding, for every production rule in P_1, a unique symbol to the set of variables:

$V_N = \{ \quad S, A, B, D, B_1, B_2, B_3, B_4, B_5 \quad \}.$

For each production $A \to w$ $(U; T)$ in P_1, we add the productions $A \to B_i$ and $B_i \to w$ to P_2, where B_i is one of the symbols added to the variables (B_i is unique for each pair of production rules in P_2). We then add the appropriate clause to the temporal restriction. For example, for the rule $S \to AA, (\emptyset; \{B, D\})$ in P_1, the rules $S \to B_1$ and $B_1 \to AA$ are added to P_2, while $(B = 0) \wedge (D = 0) \wedge X(B_1 > 0) \wedge XX(B_1 = 0)$ is added to the temporal restriction. This implies that if neither B nor D appears in the string, the rules $S \to B_1$ and $B_1 \to AA$ may be applied. Since random context grammars allow cases where more than one production is applicable to a string, the clauses in the temporal restriction are separated by $\vee$. The sets of productions and temporal restriction are:

$P_2 = \{ \quad S \to B_1, \quad B_1 \to AA,$
$A \to B_2, \quad B_2 \to B,$
$B \to B_3, \quad B_3 \to S,$
$A \to B_4, \quad B_4 \to D,$
$D \to B_5, \quad B_5 \to a \quad \}$ and

$q = G[\quad ((B = 0) \wedge (D = 0) \wedge X(B_1 > 0) \wedge XX(B_1 = 0)) \vee$
$((S = 0) \wedge (D = 0) \wedge X(B_2 > 0) \wedge XX(B_2 = 0)) \vee$
$((A = 0) \wedge (D = 0) \wedge X(B_3 > 0) \wedge XX(B_3 = 0)) \vee$
$((S = 0) \wedge (B = 0) \wedge X(B_4 > 0) \wedge XX(B_4 = 0)) \vee$
$((S = 0) \wedge (A = 0) \wedge (B = 0) \wedge X(B_5 > 0) \wedge XX(B_5 > 0)) \quad].$

At least one of the clauses has to hold for every step of the derivation. Starting with S, the propositions $B = 0$ and $D = 0$ hold, and B_1 is produced, followed by AA. Either one of the clauses $(S = 0) \wedge (D = 0) \wedge X(B_2 > 0) \wedge XX(B_2 = 0)$ or $(S = 0) \wedge (B = 0) \wedge X(B_4 > 0) \wedge XX(B_4 = 0)$ holds next, allowing the sequence of productions $A \to B_2$, $B_2 \to B$ or $A \to B_4$, $B_4 \to D$, respectively. On closer inspection

it transpires that the application of $A \rightarrow B_4$ leads to a terminating derivation, while $A \rightarrow B_2$, followed by $B_2 \rightarrow B$ produces a string containing both A and B. The repeated application of $A \rightarrow B_2$ is forced until the string contains only B's. Similarly, after applying $B \rightarrow B_3$ and $B_3 \rightarrow S$, no production from S is allowed until no more B's are left in the string, that results in a string containing only S's. Suppose that, after a number of iterations, the string becomes S^i. By applying the same sequence of rules as before, S^i will lead to A^{2i}, that in turn leads to S^{2i}. This cycle continues until the application of $A \rightarrow B_4$ and $B_4 \rightarrow D$ results in a terminating derivation. Clearly, this grammar generates the language $\{a^{2^n} : n \geq 1\}$.

The method for simulating a random context grammar with a temporal grammar is now described formally. Suppose the random context grammar is given as $\mathcal{G}_1 = (V_{N_1}, V_{T_1}, P_1, S_1)$. We show how to construct a temporal grammar, $\mathcal{G}_2$, to generate the same language.

1. Suppose $\mathcal{G}_2 = (V_{N_2}, V_{T_2}, P_2, S_2, q)$.
2. Initially, let $V_{N_2} = V_{N_1}$, $V_{T_2} = V_{T_1}$, $P_2 = \{\}$ and $S_2 = S_1$.
3. The temporal restriction, q, has the form $G[c_1 \vee c_2 \vee \cdots \vee c_r]$, $r = \#P_1$, where $\#P_1$ denotes the number of elements in P_1 and every c_j, $1 \leq j \leq r$ is a clause.
4. Suppose we number the productions in P_1 from 1 to $\#P_1$. For the ith production $A \rightarrow w\ (U;T)$ in P_1, do the following:
   (a) Add the unique variable B_i to V_{N_2}.
   (b) Add the production rules $A \rightarrow B_i$ and $B_i \rightarrow w$ to P_2.
   (c) If $U = \{U_1, U_2, \ldots, U_m\}$ and $T = \{T_1, T_2, \ldots, T_l\}$, then add the clause c_i to q, where $c_i = (U_1 > 0) \wedge (U_2 > 0) \wedge \cdots \wedge (U_m > 0) \wedge (T_1 = 0) \wedge (T_2 = 0) \wedge \cdots \wedge (T_l = 0) \wedge X(B_i > 0) \wedge XX(B_i = 0)$. Since B_i is unique, this will simulate the application of $A \rightarrow w$, as required by the random context production rule.

Theorem 4.1 *For every random context grammar, $\mathcal{G}_1$, there is a temporal grammar, $\mathcal{G}_2$, generating the same language.*

Proof: Let the temporal grammar $\mathcal{G}_2$ be constructed using the algorithm just described. We have to prove that it simulates the random context grammar $\mathcal{G}_1$.

Let $S_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_k$ be a derivation in $\mathcal{G}_1$. We show that there is a derivation $S_0 \overset{*}{\Rightarrow} w_k$ in $\mathcal{G}_2$. This assertion is proved by induction on k, where k is the length of the derivation in $\mathcal{G}_1$.

Basis: We prove the theorem for $k = 1$, where the string consists of a single symbol, the start symbol, S_0. For the random context grammar, any rule $S_0 \rightarrow w_1, (U;T)$ may be applied, unless U contains symbols other than S_0, or T contains S_0. If the production is applied, the string w_1 is derived. This is achieved in one derivation step.

Clearly, the same derivation is possible in the temporal grammar. The rule $S_0 \rightarrow B_1$, followed by $B_1 \rightarrow w_1$, may be applied, unless the n-vector entry for a variable other than S_0 is required to be non-zero, or the n-vector entry for S_0 is required to be 0. The temporal grammar generates w_1 in two derivation steps.

Induction step: Assume the assertion is true for the derivation up to length $k-1$. We use this to prove that it is true for k. If the kth string, derived from step $k-1$ in $\mathcal{G}_1$, is $w_k = v_1 S_i v_2$, we have to prove that a temporal grammar has the ability to produce any string that may be derived from $v_1 S_i v_2$ in $\mathcal{G}_1$. We show that if a string can be derived from $v_1 S_i v_2$ in $\mathcal{G}_1$, then it can be derived in $\mathcal{G}_2$, using two derivation steps.

For the random context grammar, any rule $S_i \rightarrow v, (U;T)$ may be applied, unless U contains variables other than those in w_k, or T contains at least one of the variables in w_k. If the production is applied, the string $v_1 v v_2$ is derived. This is achieved in one derivation step.

The same derivation is possible in the temporal grammar. The rule $S_i \rightarrow B_j$, followed by $B_j \rightarrow v$, with $1 \leq j \leq \#P_1$, may be applied, unless the n-vector entry for any variable other than those in w_k is required to be non-zero, or the entry for any variable in w_k is required to be 0. The temporal grammar generates $v_1 v v_2$ in two derivation steps.□

Corollary 4.1 *Temporal grammars have (at least) the generative capacity of random context grammars.*

Proof: According to Theorem 4.1, a temporal grammar, $\mathcal{G}_2$, has the ability to simulate a random context grammar, $\mathcal{G}_1$. This implies that the language generated by $\mathcal{G}_1$ is also generated by $\mathcal{G}_2$. Since a temporal grammar exist for every random context grammar, while $L(\mathcal{G}_2) = L(\mathcal{G}_1)$ for all such grammars, temporal grammars have the ability to

generate the same class of languages as random context grammars.□

We now present a simulation of a random context grammar $\mathcal{G}_1 = (\{S, A, X, Y\}, \{a\}, P, S)$, where

$$\begin{aligned} P = \{\ & S \to A && (\emptyset;\emptyset)\\ & S \to AS && (\emptyset;\emptyset)\\ & A \to X && (\emptyset;\{X,Y,S\})\\ & A \to aY && (\{X\};\emptyset)\\ & X \to a && (\emptyset;\{A\})\\ & Y \to A && (\emptyset;\{X\})\ \}. \end{aligned}$$

A temporal grammar that simulates this language is $\mathcal{G}_2 = (\{S, A, X, Y, B_1, B_2, B_3, B_4, B_5, B_6\}, \{a\}, P, S, q)$, where

$$\begin{aligned} P = \{\ & S \to B_1 && B_1 \to A\\ & S \to B_2 && B_2 \to AS\\ & A \to B_3 && B_3 \to X\\ & A \to B_4 && B_4 \to aY\\ & X \to B_5 && B_5 \to a\\ & Y \to B_6 && B_6 \to A\ \} \text{ and} \end{aligned}$$

$$\begin{aligned} q = G[\ & (X(B_1 > 0) \wedge XX(B_1 = 0)) \vee\\ & (X(B_2 > 0) \wedge XX(B_2 = 0)) \vee\\ & ((X = 0) \wedge (Y = 0) \wedge (S = 0) \wedge X(B_3 > 0) \wedge XX(B_3 = 0)) \vee\\ & ((X > 0) \wedge X(B_4 > 0) \wedge XX(B_4 = 0)) \vee\\ & ((A = 0) \wedge X(B_5 > 0) \wedge XX(B_5 = 0)) \vee\\ & ((X = 0) \wedge X(B_6 > 0) \wedge XX(B_6 = 0))\ ]. \end{aligned}$$

The random context grammar generates the language $\{a^{n(n+1)/2} : n \geq 1\}$. We will now show that the temporal grammar generates the same language.

The derivation begins by producing a string A^i, with $i \geq 1$. Since this string does not contain the variable X, the only productions that can be applied are $A \to B_3$, followed by $B_3 \to X$. This produces the string $A^j X A^{i-j}$. Since $A > 0$ and $X > 0$, the clause $((X > 0) \wedge X(B_4 > 0) \wedge XX(B_4 = 0))$ has to hold and we replace one of the A's with B_4, followed by aY, to derive $A^j X A^{i-j-1} aY$ (the instance of A we replace is of no consequence, since all occurrences will be replaced). We need to replace all instances of A with aY before any other clause can be true, and derive the string $(aY)^j X (aY)^{i-j}$. As soon as $(A = 0)$, the clause $((A = 0) \wedge X(B_5 > 0) \wedge XX(B_5 = 0))$ has to hold and X is replaced by B_5 and then a. This derives the string $(aY)^j a (aY)^{i-j}$. Y is the only

variable in the string, and since $X = 0$, we may apply $Y \to B_6$, followed by $B_6 \to A$. As long as Y is present in the string, these are the only productions that may be applied and $(aA)^j a(aA)^{i-j}$ is derived. Clearly, this string contains one less A than the string we started with, while the length of the string increased since we added the number of a's equal to the number of A's in the original string. This will happen during each cycle, until $a^{[i(i+1)/2]}$ is derived. The grammar generates $\{a^{n(n+1)/2} : n \geq 1\}$, since i instances of A, with $i \geq 1$, derives $i(i+1)/2$ instances of a.

Whether any temporal grammar can be simulated by a random context grammar is still unclear, although it is believed that temporal grammars have a larger generative capacity than random context grammars, as we will discuss in the following section.

4.2.2 Are Temporal Grammars Stronger?

The examples of temporal grammars that we have presented all generate languages that are also generated by other grammars with regulated rewriting, e.g. random context grammars. The question arises whether there is really any difference in the generating process. In this section we show that there is. In a derivation, according to a random context grammar, the next step in the derivation always depends solely on the last sentential form derived, and not at all on the way in which it was derived. In temporal grammar derivations this is not the case, as we proceed to show.

When using a temporal grammar, the restriction q may regulate the derivation over any number of timesteps. In some cases a sentential form, w, may be part of a terminating derivation, while other derivations containing w can never terminate.

To explain, consider $\mathcal{G} = \{V_N, V_T, P, S_0, q\}$, a temporal grammar for the language $L = \{x : x \in \{a\}^*,\ |x| = 4\}$, with

$V_N = \{S_0, S_2, S_1\}$,

$V_T = \{a\}$,

$P = \{\ S_0 \to S_2S_2S_1S_1 | S_1S_1S_1S_1,$

$\quad\quad S_1 \to S_2,$

$\quad\quad S_2 \to a\ \}$ and

$q = G[\ \ (S_0 > 0) \Rightarrow XXXX(S_1 = 0)\ \ ]$.

Consider the following derivations:

Γ : $S_0 \Rightarrow S_2S_2S_1S_1 \Rightarrow S_2S_2S_2S_1 \Rightarrow S_2S_2S_2S_2 \Rightarrow aS_2S_2S_2 \Rightarrow aaS_2S_2 \Rightarrow aaaS_2$
$\Rightarrow aaaa$ and

Λ : $S_0 \Rightarrow S_1S_1S_1S_1 \Rightarrow S_2S_1S_1S_1 \Rightarrow S_2S_2S_1S_1 \Rightarrow S_2S_2S_2S_1$.

Although both Γ and Λ contain the string $S_2S_2S_1S_1$, it is impossible to find a successful way to terminate Λ, since $S_1 = 0$ can never hold within four timesteps from the start symbol if $S_0 \rightarrow S_1S_1S_1S_1$ was applied.

Unlike random context grammars, temporal grammars have the ability to consider the history of a derivation. However, there are cases where more than one derivation in a temporal grammar share a history. If one of these derivations produces a string in the language, the others have the ability to do the same, as the following theorem shows.

Theorem 4.2 *Suppose* $\Gamma : S_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_m \Rightarrow \cdots \Rightarrow \alpha$ *is a valid derivation in a grammar* $\mathcal{G}$*, with* $w_1, w_2, \ldots, w_m \in V_{\mathcal{G}}^*$ *and* $\alpha \in V_T^*$*. A derivation* $\Lambda : S_0 \Rightarrow v_1 \Rightarrow v_2 \Rightarrow \cdots \Rightarrow v_m$ *has the potential to derive a string in* $L(\mathcal{G})$ *if every* w_i *and* v_i *map to the same* n*-vector for* $i = 1, 2, \ldots, m$.

Proof: Let Γ' be the section of Γ that produces w_m. The theorem states that if the sequence of n-vectors associated with Γ' is the same as the sequence associated with Λ, then Λ can produce a string in V_T^*. Suppose it is impossible for Λ to produce such a string.

Let $x(1), x(2), ..., x(m)$ be the sequence of n-vectors associated with Γ'. Since the same sequence appears in Λ, the section of the temporal restriction that forced the derivation of w_m also applies to the derivation of v_m. Furthermore, the derivation $w_m \stackrel{*}{\Rightarrow} \alpha$ is achieved by applying a sequence of productions. By applying this sequence to v_m, the derivation $v_m \stackrel{*}{\Rightarrow} \rho$, with $\rho \in V_T^*$ is possible. This contradicts the assumption that it is impossible for Λ to produce a string in V_T^*.□

4.2.3 Limitations of Temporal Grammars

In searching for an example of a context-sensitive language that is not generated by any temporal grammar, the following consideration seems important: It is clear that, if $\Gamma : S_0 \stackrel{*}{\Rightarrow} w_1Sw_2Sw_3 \stackrel{*}{\Rightarrow} \alpha_1\beta\alpha_2\gamma\alpha_3$ is a valid derivation in a temporal grammar, $\mathcal{G}$, where β derives from the left hand S and γ from the right hand S, then there will also be a valid derivation $\Lambda : S_0 \stackrel{*}{\Rightarrow} w_1Sw_2Sw_3 \stackrel{*}{\Rightarrow} \alpha_1\gamma\alpha_2\beta\alpha_3$. In other words, as soon as two (or more) instances of a variable appear in a sentential form, the grammar has

no means of regulating what descends from which instance, except, of course, in the case where there is no choice. This motivates the conjecture that a language in which the words consist of many chunks, each of which can only appear in a specific place, might provide the type of example we are seeking. Arguing along these lines, it was conjectured that a temporal grammar may be incapable of generating the language, L, consisting of the strings:

$$
\begin{aligned}
w_0 &= [b] \\
w_1 &= [ba][ab] \\
w_2 &= [baa][aba][aab] \\
&\vdots \\
w_n &= [ba^n][aba^{n-1}]\dots[a^r ba^{n-r}]\dots[a^{n-1}ba][a^n b] \\
&\vdots
\end{aligned}
$$

However, it was found that this language can be generated by a temporal grammar, and we now proceed to describe the grammar and the generative process.

$\mathcal{G} = (V_N, V_T, P, S_0, q)$ generates L, with

$V_N = \{S_0, S_\sqsupset, S_{\sqsupset_1}, S_{\sqsupset_2}, S_{\sqsupset_3}, S_{\sqsupset_4}, S_1, S_2, S_3, S_4, S_{a_2}, S_{a_1}, S_b, S_\sqsubset\}$,

$V_T = \{a, b, [,]\}$,

$$
\begin{aligned}
P = \{\ & S_0 \to S_\sqsubset S_b S_1, && S_{a_2} \to S_{a_1}, \\
& S_1 \to S_{\sqsupset_1} S_\sqsubset S_4 S_2 | S_{\sqsupset_1}, && S_2 \to S_3, \\
& S_{\sqsupset_1} \to S_{a_1} S_{\sqsupset_2} | S_\sqsupset, && S_3 \to S_1, \\
& S_{\sqsupset_2} \to S_{\sqsupset_3}, && S_\sqsubset \to [, \\
& S_{\sqsupset_3} \to S_{\sqsupset_4}, && S_\sqsupset \to], \\
& S_{\sqsupset_4} \to S_{\sqsupset_1}, && S_{a_1} \to a, \\
& S_4 \to S_{a_2} S_4 | S_b, && S_b \to b \quad \} \text{ and}
\end{aligned}
$$

$$
\begin{aligned}
q = G[\ & ((S_1 > 0) && \Rightarrow X(S_1 = 0)) \wedge \\
& ((S_2 > 0) \wedge (S_{\sqsupset_1} > 0) && \Rightarrow X((S_{\sqsupset_3} = 0) \wedge (S_{a_2} = 0) \wedge (S_4 > 0) \wedge \\
& && \quad (S_3 = 0) \wedge (S_\sqsupset = 0))\ U\ (S_{\sqsupset_1} = 0)) \wedge \\
& ((S_2 > 0) \wedge (S_{\sqsupset_1} = 0) \wedge \\
& (S_{\sqsupset_2} > 0) \wedge (S_{\sqsupset_3} = 0) \wedge (S_{a_2} = 0) && \Rightarrow X(S_{\sqsupset_3} > 0) \wedge XX(S_{a_2} > 0) \wedge \\
& && \quad XXX(S_{\sqsupset_3} = 0) \wedge XXXX(S_{a_2} = 0)) \wedge \\
& ((S_2 > 0) \wedge (S_{\sqsupset_2} = 0) \wedge \\
& (S_{\sqsupset_3} = 0) \wedge (S_{\sqsupset_4} > 0) && \Rightarrow X(S_3 > 0)) \wedge \\
& ((S_3 > 0) \wedge (S_{\sqsupset_4} > 0) && \Rightarrow X((S_1 = 0) \wedge (S_{\sqsupset_2} = 0) \wedge (S_4 > 0) \wedge \\
& && \quad (S_{a_2} = 0) \wedge (S_\sqsupset = 0))\ U\ (S_{\sqsupset_4} = 0)) \wedge \\
& ((S_3 > 0) \wedge (S_{\sqsupset_4} = 0) && \Rightarrow X(S_4 = 0) \wedge XX(S_3 = 0)) \wedge \\
& ((S_1 = 0) \wedge (S_2 = 0) \wedge (S_3 = 0) && \Rightarrow X(S_{\sqsupset_2} = 0)\ U\ (S_{\sqsupset_1} = 0))\ ].
\end{aligned}
$$

Before continuing, note that the temporal restriction has the form $G[c_1 \wedge c_2 \wedge \cdots \wedge c_p]$, $p \geq 0$, where every $c_j, 1 \leq j \leq p$ is a clause. The grammar works according to the following principle:
The string $S_{\sqsubset}S_bS_1$, that is directly derived from S_0, has the ability to produce the first string in the language, $[b]$. In order to produce a longer string, a new *term*[1] is appended to the back of the string, using the variable S_1, after which an a (produced by S_{a_1}) is added to each term to precede every occurrence of the variable $S_{\sqsupset_1}$[2]. The new term contains a number of a's equal to the number of $S_{\sqsupset_2}$'s (finally]'s) in the existing string, followed by a b. The method will now be described in more detail by showing how the strings $[b]$ and $[ba][ab]$ are produced. Note that terminals may be produced in some cases, although the derivation does not show this.

The string $S_{\sqsubset}S_bS_1$, that follows from the start symbol, S_0, contains the variables $S_{\sqsubset}$, S_b and S_1. The clause in the temporal restriction that applies to this string is $(S_1 > 0) \Rightarrow X(S_1 = 0)$, implying that the next string is either $S_{\sqsubset}S_bS_{\sqsupset_1}S_{\sqsubset}S_4S_2$ or $S_{\sqsubset}S_bS_{\sqsupset_1}$. Examining the latter, it should be clear that the clause $(S_1 = 0) \wedge (S_2 = 0) \wedge (S_3 = 0) \Rightarrow X(S_{\sqsupset_2} = 0)\ U\ (S_{\sqsupset_1} = 0)$ has to hold, since the string does not contain S_1, S_2 or S_3. The variable $S_{\sqsupset}$ replaces $S_{\sqsupset_1}$ and finally the string $[b]$ is produced, as $S_{\sqsupset_2}$ may not appear in the string until there are no occurrences of $S_{\sqsupset_1}$.

However, should the restrictive clause yield $S_{\sqsubset}S_bS_{\sqsupset_1}S_{\sqsubset}S_4S_2$, both S_2 and $S_{\sqsupset_1}$ appear in the string and, by applying the production permitted by $(S_2 > 0) \wedge (S_{\sqsupset_1} > 0) \Rightarrow X((S_{\sqsupset_3} = 0) \wedge (S_{a_2} = 0) \wedge (S_4 > 0) \wedge (S_3 = 0) \wedge (S_{\sqsupset} = 0))\ U\ (S_{\sqsupset_1} = 0)$, the string $S_{\sqsubset}S_bS_{a_1}S_{\sqsupset_2}S_{\sqsubset}S_4S_2$ is produced [3].

Again, the clause $(S_2 > 0) \wedge (S_{\sqsupset_1} = 0) \wedge (S_{\sqsupset_2} > 0) \wedge (S_{\sqsupset_3} = 0) \wedge (S_{a_2} = 0) \Rightarrow X(S_{\sqsupset_3} > 0) \wedge XX(S_{a_2} > 0) \wedge XXX(S_{\sqsupset_3} = 0) \wedge XXXX(S_{a_2} = 0)$, has to hold and it forces the derivation to add one instance of the variable S_{a_1} to the appended term for every instance of $S_{\sqsupset_2}$ that is replaced. This is achieved by replacing one instance of $S_{\sqsupset_2}$ with $S_{\sqsupset_3}$, then adding S_{a_2} by applying the production $S_4 \rightarrow S_{a_2}S_4$. The variable $S_{\sqsupset_3}$ is then replaced by $S_{\sqsupset_4}$ and S_{a_2} by S_{a_1} to allow this restrictive clause to hold repeatedly and derive the string $S_{\sqsubset}S_bS_{a_1}S_{\sqsupset_4}S_{\sqsubset}S_{a_1}S_4S_2$.

In order to satisfy the clause $(S_2 > 0) \wedge (S_{\sqsupset_2} = 0) \wedge (S_{\sqsupset_3} = 0) \wedge (S_{\sqsupset_4} > 0) \Rightarrow X(S_3 > 0)$, the next string has to contain the variable S_3. This can only be achieved by applying the production $S_2 \rightarrow S_3$, that derives $S_{\sqsubset}S_bS_{a_1}S_{\sqsupset_4}S_{\sqsubset}S_{a_1}S_4S_3$.

[1] In this case a term is a shortest substring that appears between [].
[2] This variable finally produces the terminal].
[3] No other strings may be produced, since only one temporal restriction applies.

Following this, the clause $(S_3 > 0) \wedge (S_{\sqsupset_4} > 0) \Rightarrow X((S_1 = 0) \wedge (S_{\sqsupset_2} = 0) \wedge (S_4 > 0) \wedge (S_{a_2} = 0) \wedge (S_{\sqsupset} = 0))\ U\ (S_{\sqsupset_4} = 0)$, requires all instances of $S_{\sqsupset_4}$ to be replaced by $S_{\sqsupset_1}$. No other variable-producing productions are allowed until this requirement is satisfied. The string $S_{\sqsubset}S_bS_{a_1}S_{\sqsupset_1}S_{\sqsubset}S_{a_1}S_4S_3$ is derived.

Finally, the clause $(S_3 > 0) \wedge (S_{\sqsupset_4} = 0) \Rightarrow X(S_4 = 0) \wedge XX(S_3 = 0)$ removes the variable S_4 by applying $S_4 \rightarrow S_b$, after which S_3 is removed by applying the production $S_3 \rightarrow S_1$. Doing this, the string $S_{\sqsubset}S_bS_{a_1}S_{\sqsupset_1}S_{\sqsubset}S_{a_1}S_bS_1$ is derived.

The clause $(S_1 > 0) \Rightarrow X(S_1 = 0)$ may yield either $S_{\sqsubset}S_bS_{a_1}S_{\sqsupset_1}S_{\sqsubset}S_{a_1}S_bS_{\sqsupset_1}$, that will lead to the terminating string $[ba][ab]$, or $S_{\sqsubset}S_bS_{a_1}S_{\sqsupset_1}S_{\sqsubset}S_{a_1}S_bS_{\sqsupset_1}S_{\sqsubset}S_4S_2$, that will derive longer strings.

Next it is shown that any string at the beginning of a cycle produces the string needed to create a longer string in L. A string at the beginning of the cycle is of the form

$S_{\sqsubset}S_b(S_{a_1})^nS_{\sqsupset_1}\quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^{n-1}S_{\sqsupset_1}$
$\ldots\quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r}S_{\sqsupset_1}\quad \ldots$
$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^1S_{\sqsupset_1}\quad S_{\sqsubset}(S_{a_1})^nS_bS_1$

The temporal restriction is applied to this cycle in a similar fashion as the first cycle. The portion of the derivation that illustrates this follows, with alterations to each string indicated in boldface. Steps in the derivation are numbered and each step will be discussed shortly. The derivation will not show the cases where terminals may be produced, since this will have no effect on the resulting string.

$$
\begin{aligned}
& S_{\sqsubset}S_b(S_{a_1})^nS_{\sqsupset_1}\quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^{n-1}S_{\sqsupset_1} \\
& \quad \ldots\quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r}S_{\sqsupset_1}\quad \ldots \\
& S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^1S_{\sqsupset_1}\quad S_{\sqsubset}(S_{a_1})^nS_bS_1 \\
\\
\Rightarrow^1\quad & S_{\sqsubset}S_b(S_{a_1})^nS_{\sqsupset_1}\quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^{n-1}S_{\sqsupset_1} \\
& \quad \ldots\quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r}S_{\sqsupset_1}\quad \ldots \\
& S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^1S_{\sqsupset_1}\quad S_{\sqsubset}(S_{a_1})^nS_b\boldsymbol{S_{\sqsupset_1}}\quad \boldsymbol{S_{\sqsubset}S_4S_2} \\
\\
\overset{*}{\Rightarrow}^2\quad & S_{\sqsubset}S_b(S_{a_1})^n\boldsymbol{S_{a_1}S_{\sqsupset_2}}\quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^{n-1}\boldsymbol{S_{a_1}S_{\sqsupset_2}} \\
& \quad \ldots\quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r}\boldsymbol{S_{a_1}S_{\sqsupset_2}}\quad \ldots \\
& S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^1\boldsymbol{S_{a_1}S_{\sqsupset_2}}\quad S_{\sqsubset}(S_{a_1})^nS_b\boldsymbol{S_{a_1}S_{\sqsupset_2}}\quad S_{\sqsubset}S_4S_2
\end{aligned}
$$

$$\Rightarrow^{3} \quad S_{\sqsubset}S_b(S_{a_1})^{n+1}\boldsymbol{S_{\sqsupset_3}} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^nS_{\sqsupset_2}$$
$$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}S_{\sqsupset_2} \quad \ldots$$
$$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2S_{\sqsupset_2} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1S_{\sqsupset_2} \quad S_{\sqsubset}S_4S_2$$

$$\Rightarrow^{4} \quad S_{\sqsubset}S_b(S_{a_1})^{n+1}S_{\sqsupset_3} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^nS_{\sqsupset_2}$$
$$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}S_{\sqsupset_2} \quad \ldots$$
$$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2S_{\sqsupset_2} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1S_{\sqsupset_2} \quad S_{\sqsubset}\boldsymbol{S_{a_2}}S_4S_2$$

$$\Rightarrow^{5} \quad S_{\sqsubset}S_b(S_{a_1})^{n+1}\boldsymbol{S_{\sqsupset_4}} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^nS_{\sqsupset_2}$$
$$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}S_{\sqsupset_2} \quad \ldots$$
$$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2S_{\sqsupset_2} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1S_{\sqsupset_2} \quad S_{\sqsubset}S_{a_2}S_4S_2$$

$$\Rightarrow^{6} \quad S_{\sqsubset}S_b(S_{a_1})^{n+1}S_{\sqsupset_4} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^nS_{\sqsupset_2}$$
$$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}S_{\sqsupset_2} \quad \ldots$$
$$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2S_{\sqsupset_2} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1S_{\sqsupset_2} \quad S_{\sqsubset}\boldsymbol{S_{a_1}}S_4S_2$$

$$\overset{*}{\Rightarrow}^{7} \quad S_{\sqsubset}S_b(S_{a_1})^{n+1}S_{\sqsupset_4} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^n\boldsymbol{S_{\sqsupset_4}}$$
$$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}\boldsymbol{S_{\sqsupset_4}} \quad \ldots$$
$$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2\boldsymbol{S_{\sqsupset_4}} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1\boldsymbol{S_{\sqsupset_4}} \quad S_{\sqsubset}\boldsymbol{(S_{a_1})^{n+1}}S_4S_2$$

$$\Rightarrow^{8} \quad S_{\sqsubset}S_b(S_{a_1})^{n+1}S_{\sqsupset_4} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^nS_{\sqsupset_4}$$
$$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}S_{\sqsupset_4} \quad \ldots$$
$$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2S_{\sqsupset_4} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1S_{\sqsupset_4} \quad S_{\sqsubset}(S_{a_1})^{n+1}S_4\boldsymbol{S_3}$$

$$\overset{*}{\Rightarrow}^{9} \quad S_{\sqsubset}S_b(S_{a_1})^{n+1}\boldsymbol{S_{\sqsupset_1}} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^n\boldsymbol{S_{\sqsupset_1}}$$
$$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}\boldsymbol{S_{\sqsupset_1}} \quad \ldots$$
$$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2\boldsymbol{S_{\sqsupset_1}} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1\boldsymbol{S_{\sqsupset_1}} \quad S_{\sqsubset}(S_{a_1})^{n+1}S_4S_3$$

$$\Rightarrow^{10} \quad S_{\sqsubset}S_b(S_{a_1})^{n+1}S_{\sqsupset_1} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^nS_{\sqsupset_1}$$
$$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}S_{\sqsupset_1} \quad \ldots$$
$$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2S_{\sqsupset_1} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1S_{\sqsupset_1} \quad S_{\sqsubset}(S_{a_1})^{n+1}\boldsymbol{S_b}S_3$$

$$\Rightarrow^{11} \quad S_{\sqsubset}S_b(S_{a_1})^{n+1}S_{\sqsupset_1} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^nS_{\sqsupset_1}$$
$$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}S_{\sqsupset_1} \quad \ldots$$
$$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2S_{\sqsupset_1} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1S_{\sqsupset_1} \quad S_{\sqsubset}(S_{a_1})^{n+1}S_b\boldsymbol{S_1}$$

The steps in the derivation will now be discussed.

1. The clause $(S_1 > 0) \Rightarrow X(S_1 = 0)$ requires S_1 to be removed from the string. This may be done by replacing S_1 with $S_{\sqsupset_1} S_{\sqsubset} S_4 S_2$, as shown in the derivation, or by replacing S_1 with $S_{\sqsupset_1}$. The latter will finally produce a string in the language, $[1(0)^n][(0)^1 1(0)^{n-1}] \ldots [(0)^r 1(0)^{n-r}] \ldots [(0)^{n-1} 1(0)^1][(0)^n 1]$, since the clause $(S_1 = 0) \wedge (S_2 = 0) \wedge (S_3 = 0) \Rightarrow X(S_{\sqsupset_2} = 0)\ U\ (S_{\sqsupset_1} = 0)$ will force the derivation to terminate. However, the given derivation shows how a longer string in L is derived and S_1 is replaced with $S_{\sqsupset_1} S_{\sqsubset} S_4 S_2$.

2. This step represents the derivation steps needed to replace all instances of the variable $S_{\sqsupset_1}$ with $S_{a_1} S_{\sqsupset_2}$ through the repeated application of the production $S_{\sqsupset_1} \rightarrow S_{a_1} S_{\sqsupset_2}$. The clause $(S_2 > 0) \wedge (S_{\sqsupset_1} > 0) \Rightarrow X((S_{\sqsupset_3} = 0) \wedge (S_{a_2} = 0) \wedge (S_4 > 0) \wedge (S_3 = 0) \wedge (S_{\sqsupset} = 0))\ U\ (S_{\sqsupset_1} = 0)$ implies that, until the string contains no instances of $S_{\sqsupset_1}$, the variable $S_{\sqsupset_2}$ may appear, but may not be replaced by $S_{\sqsupset_3}$.

3, 4, 5 and 6. This is the beginning of the counting process to determine the number of a-producing variables in the appended term. The clause $(S_2 > 0) \wedge (S_{\sqsupset_1} = 0) \wedge (S_{\sqsupset_2} > 0) \wedge (S_{\sqsupset_3} = 0) \wedge (S_{a_2} = 0) \Rightarrow X(S_{\sqsupset_3} > 0) \wedge XX(S_{a_2} > 0) \wedge XXX(S_{\sqsupset_3} = 0) \wedge XXXX(S_{a_2} = 0)$ requires the next string to contain $S_{\sqsupset_3}$. The production $S_{\sqsupset_2} \rightarrow S_{\sqsupset_3}$ produces this symbol and may be applied. The current string contains more than one instance of the variable $S_{\sqsupset_3}$, but since these are used for counting, any such variable may be replaced, as the others will be replaced in a similar fashion. In the illustrative derivation, the leftmost $S_{\sqsupset_3}$ is replaced. After this, the restriction requires S_{a_2} to appear in the string, that is achieved by applying $S_4 \rightarrow S_{a_2} S_4$. The two steps that follow remove $S_{\sqsupset_3}$ and S_{a_2} respectively by applying $S_{\sqsupset_3} \rightarrow S_{\sqsupset_4}$ and $S_{a_2} \rightarrow S_{a_1}$. The first a-producing variable has now been added to the appended term.

7. The restrictive clause $(S_2 > 0) \wedge (S_{\sqsupset_1} = 0) \wedge (S_{\sqsupset_2} > 0) \wedge (S_{\sqsupset_3} = 0) \wedge (S_{a_2} = 0) \Rightarrow X(S_{\sqsupset_3} > 0) \wedge XX(S_{a_2} > 0) \wedge XXX(S_{\sqsupset_3} = 0) \wedge XXXX(S_{a_2} = 0)$ forces all instances of $S_{\sqsupset_2}$ to be replaced. For every replacement, one S_{a_1} is added to the last term.

8. The production $S_2 \rightarrow S_3$ replaces S_2 with S_3, as required by $(S_2 > 0) \wedge (S_{\sqsupset_2} = 0) \wedge (S_{\sqsupset_3} = 0) \wedge (S_{\sqsupset_4} > 0) \Rightarrow X(S_3 > 0)$.

9. Once S_3 appears in the string together with at least one instance of $S_{\sqsupset_4}$, the clause $(S_3 > 0) \wedge (S_{\sqsupset_4} > 0) \Rightarrow X((S_1 = 0) \wedge (S_{\sqsupset_2} = 0) \wedge (S_4 > 0) \wedge (S_{a_2} = 0) \wedge (S_{\sqsupset} = 0))\ U\ (S_{\sqsupset_4} = 0)$ requires every S_3 to be replaced by S_1, before any other variable-producing production is legal.

10 and 11. Lastly, $(S_3 > 0) \wedge (S_{\sqsupset_4} = 0) \Rightarrow X(S_4 = 0) \wedge XX(S_3 = 0)$ requires S_4 to be removed, that is done by applying $S_4 \rightarrow S_b$. Following this, the restriction also states that S_3 should be removed from the string and $S_3 \rightarrow S_1$ is applied.

After completing the cycle, the string
$S_{\sqsubset}S_b(S_{a_1})^{n+1}S_{\sqsupset_1} \quad S_{\sqsubset}(S_{a_1})^1S_b(S_{a_1})^nS_{\sqsupset_1}$
$\ldots \quad S_{\sqsubset}(S_{a_1})^rS_b(S_{a_1})^{n-r+1}S_{\sqsupset_1} \quad \ldots$
$S_{\sqsubset}(S_{a_1})^{n-1}S_b(S_{a_1})^2S_{\sqsupset_1} \quad S_{\sqsubset}(S_{a_1})^nS_b(S_{a_1})^1S_{\sqsupset_1} \quad S_{\sqsubset}(S_{a_1})^{n+1}S_bS_1$
can be used to derive a longer string, or produce $[b(a)^{n+1}][(a)^1b(a)^n]\ldots[(a)^rb(a)^{n-r+1}]$ $\ldots[(a)^{n-1}b(a)^2][(a)^nb(a)^1][(a)^{n+1}b]$, that is in L.

All the strings in the language generated by the grammar we just described have the same form. To find a language that cannot be generated by temporal grammars, it may be necessary to consider sets of strings in which the form of some of the strings deviate from others. Again consider the strings

$$\begin{aligned} w_0 &= [b] \\ w_1 &= [ba][ab] \\ w_2 &= [baa][aba][aab] \\ &\vdots \\ w_n &= [ba^n][aba^{n-1}]\ldots[a^rba^{n-r}]\ldots[a^{n-1}ba][a^nb] \\ &\vdots \end{aligned}$$

Suppose we alter w_i whenever i is a prime number, by swapping the first and last term in the string. For example, the string $w_3 = [baaa][abaa][aaba][aaab]$ is changed to $w_3 = [aaab][abaa][aaba][baaa]$. Since it is believed that a temporal grammar may not be able to determine whether i is prime, such a grammar may not have the ability to generate this language.

Chapter 5

Conclusion

The primary goal of this thesis was to design and formulate a method to regulate the derivations in a grammar by introducing temporal logic as a regulating device. As mentioned in Chapter 1, other goals included examining the properties, as well as the generative ability of this grammar.

In Section 5.1 of this chapter we present a brief overview on some of the properties of temporal grammars and remark on these properties, as well as the generative power of the grammars. Possible future work is discussed in Section 5.2, where we also present some concluding remarks.

5.1 Remarks on Temporal Grammars

The design and formulation of temporal grammars were achieved by showing how a temporal structure can be associated with a grammar and by defining temporal grammars in Section 4.1.3. Using these grammars to generate commonly used examples of non-context-free languages showed that they have the ability to generate these languages using context-free production rules.

In general, the temporal restrictions discussed in this thesis were designed to control derivations in such a way that blocked derivations were avoided. However, including an atomic proposition such as S_i *is prime*, together with the restriction $F(S_i$ *is prime*$)$ may be able to generate the language $\{a^n | n$ *is prime*$\}$, but will not guide the derivation in order to achieve this.

When we compared temporal grammars to random context grammars in Section 4.2, it was found that temporal grammars are at least as powerful as random context grammars. Since there exists an equivalence between random context, programmed and matrix grammars, temporal grammars are not less powerful than matrix and programmed grammars either.

5.2 Future Work and Concluding Remarks

A challenging problem is to determine whether temporal grammars may have a greater generative capacity than random context grammars, as well as other regulated grammars. An investigation on the generative power may also include the closure properties, as well as finding languages that cannot be generated by temporal grammars.

Another aspect to consider is that of the atomic propositions in the grammar. In this thesis the propositions PZ_i and PG_i were used to indicate the absence or presence of a variable S_i in a string (the propositions were written $S_i = 0$ and $S_i > 0$). These propositions may be altered to include numbers other than 0, such as $S = 5$ or $S > 10$. An examination on the capacity of temporal grammars with different atomic propositions may prove interesting.

Temporal grammars combine two fields in Computer Science, grammars and temporal logic. Since a derivation is a process that evolves over time, using such grammars is quite natural and may present numerous research opportunities.

Bibliography

[1] S. Abraham. Some questions of phrase-structure grammars. *Computational Linguistics*, 4:61–70, 1965.

[2] J. Dassow and G. Păun. *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, 1989.

[3] J. Dassow, G. Păun, and A. Salomaa. *Grammars with Controlled Derivations*, volume 2 of *Handbook of Formal Languages*, chapter 3, pages 101–150. Springer, 1997.

[4] E. Emerson. *Formal Models and Semantics*, chapter 16, pages 997–1072. Handbook of Theoretical Computer Science. Elsevier, 1990.

[5] I. Fris. Grammars with partial orderings of the rules. *Information and Control*, 12:415–425, 1968.

[6] S. Ginsburg and E. H. Spanier. Control sets on grammars. *Mathematical Systems Theory*, 2(2):159–177, 1968.

[7] S. Greibach and J. Hopcroft. Scattered context grammars. *Journal of Computer and System Sciences*, 3:233–247, 1969.

[8] E. Shamir J. Gonczarowski. Pattern selector grammars and several parsing algorithms in the context-free style. *Journal of Computer and System Sciences*, 30:249–273, 1985.

[9] H.C.M. Klein and G. Rozenberg. Context-free like restrictions on selective rewriting. *Theoretical Computer Science*, 16:237–269, 1981.

[10] O. Mayer. Some restrictive devices for context-free grammars. *Information and Control*, 20:69–92, 1972.

[11] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the Association for Computing Machinery*, 16(1):107–131, 1969.

[12] A. Salomaa. On some families of formal languages obtained by regulated derivations. *Annales Academiæ Scientiarum Fennicæ*, 1970.

[13] R. Siromoney and K. Krithivasan. Parallel context-free grammar. *Information and Control*, 24:155–162, 1974.

[14] A.P.J. van der Walt. Random context languages. In *Information Processing*, 1972.